Read the Docs Documentation

Release 7.0.0

Eric Holscher, Charlie Leifer, Bobby Grace

Jan 18, 2022

GENERAL

Documentation for running your own local version of Read the Docs for development, or taking the open source Read the Docs codebase for your own custom installation.

CHAPTER

ONE

CONTRIBUTING TO READ THE DOCS

You are here to help on Read the Docs? Awesome, feel welcome and read the following sections in order to know how to ask questions and how to work on something.

All members of our community are expected to follow our *Code of Conduct*. Please make sure you are welcoming and friendly in all of our spaces.

1.1 Get in touch

- Ask usage questions ("How do I?") on StackOverflow.
- Report bugs, suggest features or view the source code on GitHub.
- Discuss topics on Gitter.

1.2 Contributing to development

If you want to deep dive and help out with development on Read the Docs, then first get the project installed locally according to the *installation guide*. After that is done we suggest you have a look at tickets in our issue tracker that are labelled Good First Issue. These are meant to be a great way to get a smooth start and won't put you in front of the most complex parts of the system.

If you are up to more challenging tasks with a bigger scope, then there are a set of tickets with a Feature or Improvement tag. These tickets have a general overview and description of the work required to finish. If you want to start somewhere, this would be a good place to start (make sure that the issue also have the Accepted label). That said, these aren't necessarily the easiest tickets. They are simply things that are explained. If you still didn't find something to work on, search for the Sprintable label. Those tickets are meant to be standalone and can be worked on ad-hoc.

You can read all of our *Read the Docs developer documentation* to understand more the development of Read the Docs. When contributing code, then please follow the standard Contribution Guidelines set forth at contribution-guide.org.

1.3 Contributing to documentation

Documentation for Read the Docs itself is hosted by Read the Docs at https://docs.readthedocs.io (likely the website you are currently reading).

There are guidelines around writing and formatting documentation for the project. For full details, including how to build it, see *Building and Contributing to Documentation*.

1.4 Triaging tickets

Here is a brief explanation on how we triage incoming tickets to get a better sense of what needs to be done on what end.

Note: You will need Triage permission on the project in order to do this. You can ask one of the members of the Team to give you access.

1.4.1 Initial triage

When sitting down to do some triaging work, we start with the list of untriaged tickets. We consider all tickets that do not have a label as untriaged. The first step is to categorize the ticket into one of the following categories and either close the ticket or assign an appropriate label. The reported issue …

… is not valid If you think the ticket is invalid comment why you think it is invalid, then close the ticket. Tickets might be invalid if they were already fixed in the past or it was decided that the proposed feature will not be implemented because it does not conform with the overall goal of Read the Docs. Also if you happen to know that the problem was already reported, reference the other ticket that is already addressing the problem and close the duplicate.

Examples:

- *Builds fail when using matplotlib*: If the described issue was already fixed, then explain and instruct to re-trigger the build.
- *Provide way to upload arbitrary HTML files*: It was already decided that Read the Docs is not a dull hosting platform for HTML. So explain this and close the ticket.

… does not provide enough information Add the label **Needed: more information** if the reported issue does not contain enough information to decide if it is valid or not and ask on the ticket for the required information to go forward. We will re-triage all tickets that have the label **Needed: more information** assigned. If the original reporter left new information we can try to re-categorize the ticket. If the reporter did not come back to provide more required information after a long enough time, we will close the ticket (this will be roughly about two weeks).

Examples:

- *My builds stopped working. Please help!* Ask for a link to the build log and for which project is affected.

… is a valid feature proposal If the ticket contains a feature that aligns with the goals of Read the Docs, then add the label **Feature**. If the proposal seems valid but requires further discussion between core contributors because there might be different possibilities on how to implement the feature, then also add the label **Needed: design decision**.

Examples:

- *Provide better integration with service XYZ*

- *Achieve world domination* (also needs the label **Needed: design decision**)

… is a small change to the source code If the ticket is about code cleanup or small changes to existing features would likely have the **Improvement** label. The distinction for this label is that these issues have a lower priority than a Bug, and aren't implementing new features.

Examples:

- *Refactor namedtuples to dataclasess*
- *Change font size for the project's title*

… is a valid problem within the code base: If it's a valid bug, then add the label **Bug**. Try to reference related issues if you come across any.

Examples:

- *Builds fail if conf.py contains non-ascii letters*

… is a currently valid problem with the infrastructure: Users might report about web server downtimes or that builds are not triggered. If the ticket needs investigation on the servers, then add the label **Operations**.

Examples:

- *Builds are not starting*

… is a question and needs answering: If the ticket contains a question about the Read the Docs platform or the code, then add the label **Support**.

Examples:

- *My account was set inactive. Why?*
- *How to use C modules with Sphinx autodoc?*
- *Why are my builds failing?*

… requires a one-time action on the server: Tasks that require a one time action on the server should be assigned the two labels **Support** and **Operations**.

Examples:

- *Please change my username*
- *Please set me as owner of this abandoned project*

After we finished the initial triaging of new tickets, no ticket should be left without a label.

1.4.2 Additional labels for categorization

Additionally to the labels already involved in the section above, we have a few more at hand to further categorize issues.

High Priority If the issue is urgent, assign this label. In the best case also go forward to resolve the ticket yourself as soon as possible.

Good First Issue This label marks tickets that are easy to get started with. The ticket should be ideal for beginners to dive into the code base. Better is if the fix for the issue only involves touching one part of the code.

Sprintable Sprintable are all tickets that have the right amount of scope to be handled during a sprint. They are very focused and encapsulated.

For a full list of available labels and their meanings, see *Overview of issue labels*.

1.4.3 Helpful links for triaging

Here is a list of links for contributors that look for work:

- Untriaged tickets: Go and triage them!
- Tickets labelled with Needed: more information: Come back to these tickets once in a while and close those that did not get any new information from the reporter. If new information is available, go and re-triage the ticket.
- Tickets labelled with Operations: These tickets are for contributors who have access to the servers.
- Tickets labelled with Support: Experienced contributors or community members with a broad knowledge about the project should handle those.
- Tickets labelled with Needed: design decision: Project leaders must take actions on these tickets. Otherwise no other contributor can go forward on them.

1.5 Helping on translations

If you wish to contribute translations, please do so on Transifex.

CHAPTER

TWO

CODE OF CONDUCT

Like the technical community as a whole, the Read the Docs team and community is made up of a mixture of professionals and volunteers from all over the world, working on every aspect of the mission - including mentorship, teaching, and connecting people.

Diversity is one of our huge strengths, but it can also lead to communication issues and unhappiness. To that end, we have a few ground rules that we ask people to adhere to. This code applies equally to founders, mentors and those seeking help and guidance.

This isn't an exhaustive list of things that you can't do. Rather, take it in the spirit in which it's intended - a guide to make it easier to enrich all of us and the technical communities in which we participate.

This code of conduct applies to all spaces managed by the Read the Docs project. This includes live chat, mailing lists, the issue tracker, and any other forums created by the project team which the community uses for communication. In addition, violations of this code outside these spaces may affect a person's ability to participate within them.

If you believe someone is violating the code of conduct, we ask that you report it by emailing dev@readthedocs.org.

- **Be friendly and patient.**
- **Be welcoming.** We strive to be a community that welcomes and supports people of all backgrounds and identities. This includes, but is not limited to members of any race, ethnicity, culture, national origin, colour, immigration status, social and economic class, educational level, sex, sexual orientation, gender identity and expression, age, size, family status, political belief, religion, and mental and physical ability.
- **Be considerate.** Your work will be used by other people, and you in turn will depend on the work of others. Any decision you take will affect users and colleagues, and you should take those consequences into account when making decisions. Remember that we're a world-wide community, so you might not be communicating in someone else's primary language.
- **Be respectful.** Not all of us will agree all the time, but disagreement is no excuse for poor behavior and poor manners. We might all experience some frustration now and then, but we cannot allow that frustration to turn into a personal attack. It's important to remember that a community where people feel uncomfortable or threatened is not a productive one. Members of the Read the Docs community should be respectful when dealing with other members as well as with people outside the Read the Docs community.
- **Be careful in the words that you choose.** We are a community of professionals, and we conduct ourselves professionally. Be kind to others. Do not insult or put down other participants. Harassment and other exclusionary behavior aren't acceptable. This includes, but is not limited to:
  - Violent threats or language directed against another person.
  - Discriminatory jokes and language.
  - Posting sexually explicit or violent material.
  - Posting (or threatening to post) other people's personally identifying information ("doxing").
  - Personal insults, especially those using racist or sexist terms.

    - Unwelcome sexual attention.
    - Advocating for, or encouraging, any of the above behavior.
    - Repeated harassment of others. In general, if someone asks you to stop, then stop.

- **When we disagree, try to understand why.** Disagreements, both social and technical, happen all the time and Read the Docs is no exception. It is important that we resolve disagreements and differing views constructively. Remember that we're different. The strength of Read the Docs comes from its varied community, people from a wide range of backgrounds. Different people have different perspectives on issues. Being unable to understand why someone holds a viewpoint doesn't mean that they're wrong. Don't forget that it is human to err and blaming each other doesn't get us anywhere. Instead, focus on helping to resolve issues and learning from mistakes.

Original text courtesy of the Speak Up! project. This version was adopted from the Django Code of Conduct.

CHAPTER

THREE

OVERVIEW OF ISSUE LABELS

Here is a full list of labels that we use in the GitHub issue tracker and what they stand for.

Accepted Issues with this label are issues that the core team has accepted on to the roadmap. The core team focuses on accepted bugs, features, and improvements that are on our immediate roadmap and will give priority to these issues. Pull requests could be delayed or closed if the pull request doesn't align with our current roadmap. An issue or pull request that has not been accepted should either eventually move to an accepted state, or should be closed. As an issue is accepted, we will find room for it on our roadmap, either on an upcoming release (point release milestones), or on a future milestone project (named milestones).

Bug An issue describing unexpected or malicious behaviour of the readthedocs.org software. A Bug issue differs from an Improvement issue in that Bug issues are given priority on our roadmap. On release, these issues generally only warrant incrementing the patch level version.

Design Issues related to the UI of the readthedocs.org website.

Feature Issues that describe new features. Issues that do not describe new features, such as code cleanup or fixes that are not related to a bug, should probably be given the Improvement label instead. On release, issues with the Feature label warrant at least a minor version increase.

Good First Issue This label marks issues that are easy to get started with. The issue should be ideal for beginners to dive into the code base.

Priority: high Issues with this label should be resolved as quickly as possible.

Priority: low Issues with this label won't have the immediate focus of the core team.

Improvement An issue with this label is not a Bug nor a Feature. Code cleanup or small changes to existing features would likely have this label. The distinction for this label is that these issues have a lower priority on our roadmap compared to issues labeled Bug, and aren't implementing new features, such as a Feature issue might.

Needed: design decision Issues that need a design decision are blocked for development until a project leader clarifies the way in which the issue should be approached.

Needed: documentation If an issue involves creating or refining documentation, this label will be assigned.

Needed: more information This label indicates that a reply with more information is required from the bug reporter. If no response is given by the reporter, the issue is considered invalid after 2 weeks and will be closed. See the documentation about our *triage process* for more information.

Needed: patch This label indicates that a patch is required in order to resolve the issue. A fix should be proposed via a pull request on GitHub.

Needed: tests This label indicates that a better test coverage is required to resolve the issue. New tests should be proposed via a pull request on GitHub.

Needed: replication This label indicates that a bug has been reported, but has not been successfully replicated by another user or contributor yet.

Operations Issues that require changes in the server infrastructure.

PR: work in progress Pull Requests that are not complete yet. A final review is not possible yet, but every Pull Request is open for discussion.

PR: hotfix Pull request was applied directly to production after a release. These pull requests still need review to be merged into the next release.

Sprintable Sprintable are all issues that have the right amount of scope to be handled during a sprint. They are very focused and encapsulated.

Status: blocked The issue cannot be resolved until some other issue has been closed. See the issue's log for which issue is blocking this issue.

Status: stale A issue is stale if it there has been no activity on it for 90 days. Once a issue is determined to be stale, it will be closed after 2 weeks unless there is activity on the issue.

Support Questions that needs answering but do not require code changes or issues that only require a one time action on the server will have this label. See the documentation about our *triage process* for more information.

CHAPTER

FOUR

ROADMAP

4.1 Process

Read the Docs has adopted the following workflow with regards to how we prioritize our development efforts and where the core development team focuses its time.

4.1.1 Triaging issues

Much of this is already covered in our guide on *Contributing to Read the Docs*, however to summarize the important pieces:

- New issues coming in will be triaged, but won't yet be considered part of our roadmap.
- If the issue is a valid bug, it will be assigned the `Accepted` label and will be prioritized, likely on an upcoming point release.
- If the issues is a feature or improvement, the issue might go through a design decision phase before being accepted and assigned to a milestone. This is a good time to discuss how to address the problem technically. Skipping this phase might result in your PR being blocked, sent back to design decision, or perhaps even discarded. It's best to be active here before submitting a PR for a feature or improvement.
- The core team will only work on accepted issues, and will give PR review priority to accepted issues. Pull requests addressing issues that are not on our roadmap are welcome, but we cannot guarantee review response, even for small or easy to review pull requests.

4.1.2 Milestones

We maintain two types of milestones: point release milestones for our upcoming releases, and group milestones, for blocks of work that have priority in the future.

Generally there are 2 or 3 point release milestones lined up. These point releases dictate the issues that core team has discussed as priority already. Core team should not focus on issues outside these milestones as that implies either the issue was not discussed as a priority, or the issue isn't a priority.

We follow semantic versioning for our release numbering and our point release milestones follow these guidelines. For example, our next release milestones might be `2.8`, `2.9`, and `3.0`. Releases `2.8` and `2.9` will contain bug fix issues and one backwards compatible feature (this dictates the change in minor version). Release `3.0` will contain bugfixes and at least one backwards incompatible change.

Point release milestones try to remain static, but can shift upwards on a release that included an unexpected feature addition. Sometimes the resulting PR unexpectedly includes changes that dictate a minor version increment though, according to semantic versioning. In this case, the current milestone is closed, future milestones are increased a minor point if necessary, and the remaining milestone issues are migrated to a new milestone for the next upcoming release number.

Group milestones are blocks of work that will have priority in the future, but aren't included on point releases yet. When the core team does decide to make these milestones a priority, they will be moved into point release milestones.

4.1.3 Where to contribute

It's best to pick off an issue from our current point release or group milestones, to ensure your pull request gets attention. You can also feel free to contribute on our Cleanup or Refactoring milestones. Though not a development priority, these issues are generally discrete, easier to jump into than feature development, and we especially appreciate outside contributions here as these milestones are not a place the core team can justify spending time in development currently.

4.2 Current roadmap

In addition to the point release milestones currently established, our current roadmap priorities also include:

Admin UX https://github.com/readthedocs/readthedocs.org/milestone/16

Search Improvements https://github.com/readthedocs/readthedocs.org/milestone/23

YAML File Completion https://github.com/readthedocs/readthedocs.org/milestone/28

There are also several milestones that are explicitly *not* a priority for the core team:

Cleanup https://github.com/readthedocs/readthedocs.org/milestone/10

Refactoring https://github.com/readthedocs/readthedocs.org/milestone/34

Core team will not be focusing their time on these milestones in development.

CHAPTER FIVE

DESIGN DOCUMENTS

This is where we outline the design of major parts of our project. Generally this is only available for features that have been build in the recent past, but we hope to write more of them over time.

Warning: These documents may not match the final implementation, or may be out of date.

5.1 API v3 Design Document

This document describes the design, some decisions already made and built (current Version 1 of APIv3) and an implementation plan for next Versions of APIv3.

APIv3 will be designed to be easy to use and useful to perform read and write operations as the main two goals.

It will be based on Resources as APIv2 but considering the `Project` resource as the main one, from where most of the endpoint will be based on it.

5.1.1 Goals

- Easy to use for our users (access most of resources by `slug`)
- Useful to perform read and write operations
- Authentication/Authorization
  - Authentication based on scoped-tokens
  - Handle Authorization nicely using an abstraction layer
- Cover most useful cases:
  - Integration on CI (check build status, trigger new build, etc)

  - Usage from public Sphinx/MkDocs extensions
  - Allow creation of flyout menu client-side
  - Simplify migration from other services (import projects, create multiple redirects, etc)

5.1.2 Non-Goals

- Filter by arbitrary and useless fields
  - "Builds with `exit_code=1`"
  - "Builds containing `ERROR` on their output"
  - "Projects created after X datetime"
  - "Versions with tag `python`"
- Cover *all the actions* available from the WebUI

5.1.3 Problems with APIv2

There are several problem with our current APIv2 that we can list:

- No authentication
- It's read-only
- Not designed for slugs
- Useful APIs not exposed (only for internal usage currently)
- Error reporting is a mess
- Relationships between API resources is not obvious
- Footer API endpoint returns HTML

5.1.4 Implementation stages

Version 1

The first implementation of APIv3 will cover the following aspects:

- Authentication
  - all endpoints require authentication via `Authorization:` request header
  - detail endpoints are available for all authenticated users
  - only Project's maintainers can access listing endpoints
  - personalized listing
- Read and Write
  - edit attributes from Version (only `active` and `privacy_level`)
  - trigger Build for a specific Version
- Accessible by slug
  - Projects are accessed by `slug`

  - Versions are accessed by `slug`
  - `/projects/` endpoint is the main one and all of the other are nested under it
  - Builds are accessed by `id`, as exception to this rule
  - access all (active/non-active) Versions of a Project by `slug`
  - get latest Build for a Project (and Version) by `slug`
  - filter by relevant fields
- Proper status codes to report errors
- Browse-able endpoints
  - browse is allowed hitting `/api/v3/projects/` as starting point
  - ability to navigate clicking on other resources under `_links` attribute
- Rate limited

Version 2

Note: This is currently implemented and live.

Second iteration will polish issues found from the first step, and add new endpoints to allow *import a project and configure it* without the needed of using the WebUI as a main goal.

After Version 2 is deployed, we will invite users that reach us as beta testers to receive more feedback and continue improving it by supporting more use cases.

This iteration will include:

- Minor changes to fields returned in the objects
- Import Project endpoint
- Edit Project attributes (“Settings” and “Advanced settings-Global settings” in the WebUI)
- Trigger Build for default version
- Allow CRUD for Redirect, Environment Variables and Notifications (`WebHook` and `EmailHook`)
- Create/Delete a Project as subproject of another Project
- Documentation

Version 3

Third iteration will implement granular permissions. Keeping in mind how Sphinx extension will use it:

- `sphinx-version-warning` needs to get *all active Versions of a Project*
- An extension that creates a landing page, will need *all the subprojects of a Project*

To fulfill these requirements, this iteration will include:

- Scope-based authorization token

Version 4

- Specific endpoint for our flyout menu (returning JSON instead of HTML)

5.1.5 Out of roadmap

These are some features that we may want to build at some point. Although, they are currently out of our near roadmap because they don't affect too many users, or are for internal usage only.

- CRUD for Domain
- Add User as maintainer
- Give access to a documentation page (`objects.inv`, `/design/core.html`)
- Internal Build process

5.1.6 Nice to have

- `Request-ID` header
- JSON minified by default (maybe with `?pretty=true`)
- JSON schema and validation with docs

5.2 Build Images

This document describes how Read the Docs uses the Docker Images and how they are named. Besides, it proposes a path forward about a new way to create, name and use our Docker build images to reduce its complexity and support installation of other languages (e.g. nodejs, rust, go) as extra requirements.

5.2.1 Introduction

We use Docker images to build user's documentation. Each time a build is triggered, one of our VMs picks the task and go through different steps:

1. run some application code to spin up a Docker image into a container
2. execute `git` inside the container to clone the repository
3. analyze and parse files (`.readthedocs.yaml`) from the repository *outside* the container
4. spin up a new Docker container based on the config file
5. create the environment and install docs' dependencies inside the container
6. execute build commands inside the container
7. push the output generated by build commands to the storage

All those steps depends on specific commands versions: `git`, `python`, `virtualenv`, `conda`, etc. Currently, we are pinning only a few of them in our Docker images and that have caused issues when re-deploying these images with bugfixes: **the images are not reproducible over time**.

Note: We have been improving the reproducibility of our images by adding some tests cases. These are run inside the Docker image after it's built and check that it contains the versions we expect.

To allow users to pin the image we ended up exposing three images: `stable`, `latest` and `testing`. With that naming, we were able to bugfix issues and add more features on each image without asking the users to change the image selected in their config file.

Then, when a completely different image appeared and after testing `testing` image enough, we discarded `stable`, old `latest` became the new `stable` and old `testing` became the new `latest`. This produced issues to people pinning their images to any of these names because after this change, *we changed all the images for all the users* and many build issues arrised!

5.2.2 Goals

- release completely new Docker images without forcing users to change their pinned image (`stable`, `latest`, `testing`)
- allow users to select language requirements instead of an image name
- use a `base` image with the dependencies that don't change frequently (OS and base requirements)
- `base` image naming is tied to the OS version (e.g. Ubuntu LTS)
- allow us to add/update a Python version without affecting the `base` image
- reduce size on builder VM disks by sharing Docker image layers
- allow users to specify extra languages (e.g. nodejs, rust, go)
- de-motivate the usage of `stable`, `latest` and `testing`; and promote declaring language requirements instead
- new images won't contain old/deprecated OS (eg. Ubuntu 18) and Python versions (eg. 3.5, miniconda2)
- install language requirements *at built time* using `asdf` and its plugins
- create local mirrors for all languages supported
- deleting a pre-built image won't make builds to fail; only make them slower
- support only the latest Ubuntu LTS version and keep the previous one as long as it's officially supported

5.2.3 Non goals

- allow creation/usage of custom Docker images
- allow to execute arbitrary commands via hooks (eg. `pre_build`)
- automatically build & push *all* images on commit
- pre-built multiple images for all the languages combinations

5.2.4 Pre-built build image structure

The new pre-built images will depend only on the Ubuntu OS. They will contain all the requirements to add extra languages support at built time via `asdf` command.

- `ubuntu20-base`
  - labels
  - environment variables
  - system dependencies
  - install requirements
  - LaTeX dependencies (for PDF generation)
  - languages version manager (`asdf`) and its plugins for each language
  - UID and GID

Instead of building all the Docker image per language versions combination, it will be easier to install all of them at build time using the same steps. Installing a language only adds a few seconds when binaries are provided. However, to reduce the time to install these languages as much as possible, a local mirror hosted on S3 for each language will be created.

It's important to note that Python does not provide binaries and compiling a version takes around ~2 minutes. However, the Python versions could be pre-compiled and expose their binaries via S3 to builders. Then, at build time, the builder will only download the binary and copy it in the correct path.

Note: Depending on the demand, Read the Docs may pre-build the most common combinations of languages used by users. For example, `ubuntu20+python39+node14` or `ubuntu20+python39+node14+rust1`. However, this is seen as an optimization for the future and it's not required for this document.

5.2.5 Build steps

With this new approach, the steps followed by a builder will be:

1. run some application code to spin up the `-base` Docker image into a container
2. execute `git` inside the container to clone the repository
3. analyze and parse files (`.readthedocs.yaml`) from the repository *outside* the container
4. spin up a new Docker container **based on the Ubuntu OS specified** in the config file
5. **install all language dependencies from the cache**
6. create the environment and install docs' dependencies inside the container
7. execute build commands inside the container
8. push the output generated by build commands to the storage

The main difference with the current approach are:

- the image to spin up is selected depending on the OS version
- all language dependencies are installed at build time
- languages not offering binaries are pre-compiled by Read the Docs and stored in the cache

- miniconda/mambaforge are now managed with the same management tool (e.g. `asdf install python miniconda3-4.7.12`)

5.2.6 Specifying extra languages requirements

Different users may have different requirements. People with specific language dependencies will be able to install them by using `.readthedocs.yaml` config file. Example:

```yaml
build:
  os: ubuntu20
  languages:
    python: "3.9"  # supports "pypy3", "miniconda3" and "mambaforge"
    nodejs: "14"
    rust: "1.54"
    golang: "1.17"
```

Important highlights:

- do not treat Python language different from the others (will help us to support other non-Python doctools in the future)
- specifying `build.languages.python: "3"` will use Python version `3.x.y`, and may differ between builds
- specifying `build.languages.python: "3.9"` will use Python version `3.9.y`, and may differ between builds
- specifying `build.languages.nodejs: "14"` will use nodejs version `14.x.y`, and may differ between builds
- if no full version is declared, it will try first latest available on our cache, and then the latest on `asdf` (it has to match the first part of the version declared)
- specifying minor language versions is not allowed (e.g. `3.7.11`)
- not specifying `build.os` will make the config file parser to fail
- not specifying `build.languages` will make the config file parsing to fail (at least one is required)
- specifying only `build.languages.nodejs` and using Sphinx to build the docs, will make the build to fail (e.g. "Command not found")
- `build.image` is incompatible with `build.os` or `build.languages` and will produce an error
- `python.version` is incompatible with `build.os` or `build.languages` and will produce an error
- Ubuntu 18 will still be available via `stable` and `latest` images, but not in new ones
- only a subset (not defined yet) of `python`, `nodejs`, `rust` and `go` versions on `asdf` are available to select

Note: We are moving away from users specifying a particular Docker image. With the new approach, users will specify the languages requirements they need, and Read the Docs will decide if it will use a pre-built image or will spin up the base one and install these languages on the fly.

However, `build.image` will be still available for backward compatibility with `stable`, `latest` and `testing` but won't support the new `build.languages` config.

Note that knowing exactly what packages users are installing, could allow us to pre-build the most common combinations used images: `ubuntu20+py39+node14`.

5.2.7 Time required to install languages at build time

Testings using `time` command in ASG instances to install extra languages took these "real" times:

- `build-default`
  - python 3.9.6: 2m21.331s
  - mambaforge 4.10.1: 0m26.291s
  - miniconda3 4.7.12: 0m9.955s
  - nodejs 14.17.5: 0m5.603s
  - rust 1.54.0: 0m13.587s
  - golang 1.17: 1m30.428s
- `build-large`
  - python 3.9.6: 2m33.688s
  - mambaforge 4.10.1: 0m28.781s
  - miniconda3 4.7.12: 0m10.551s
  - nodejs 14.17.5: 0m6.136s
  - rust 1.54.0: 0m14.716s
  - golang 1.17: 1m36.470s

Note that the only one that required compilation was Python. All the others, spent 100% of its time downloading the binary. These download times are *way better from EU* with a home internet connection.

In the worst scenario: "none of the specified language version has a pre-built image", the build will require ~5 minutes to install all the language requirements. By providing *only* pre-built images with the Python version (that's the most time consuming), build times will only require ~2 minutes to install the others. However, requiring one version of each language is not a common case.

5.2.8 Cache language binaries on S3

`asdf` scripts can be altered to download the `.tar.gz` dist files from a different mirror than the official one. Read the Docs can make usage of this to create a mirror hosted locally on S3 to get faster download speeds. This will make a good improvement for languages that offer binaries: `nodejs`, `rust` and `go`:

- `nodejs` uses `NODEJS_ORG_MIRROR`: https://github.com/asdf-vm/asdf-nodejs/blob/master/lib/utils.sh#L5
- `rust` uses `RUSTUP_UPDATE_ROOT`: https://github.com/rust-lang/rustup/blob/master/rustup-init.sh#L23
- `go` has the URL hardcoded: https://github.com/kennyp/asdf-golang/blob/master/bin/download#L54

However, currently Python does not offer binaries and a different solution is needed. Python versions can be pre-compiled once and expose the output on the S3 for the builders to download and extract in the correct PATH.

Tip: Since we are building a special cache for pre-compiled Python versions, we could use the same method for all the other languages instead of creating a full mirror (many Gigabyes) This simple bash script download the language sources, compiles it and upload it to S3 without requiring a mirror. Note that it works in the same way for all the languages, not just for Python.

5.2.9 Questions

What Python versions will be pre-compiled and cached?

At start only a small subset of Python version will be pre-compiled:

- 2.7.x
- 3.7.x
- 3.8.x
- 3.9.x
- 3.10.x
- pypy3.x

How do we upgrade a Python version?

Python patch versions can be upgraded by re-compiling the new patch version and making it available in our cache. For example, if version 3.9.6 is the one available and 3.9.7 is released, *after updating our cache*:

- users specifying `build.languages.python: "3.9"` will get the 3.9.7 version
- users specifying `build.languages.python: "3"` will get the 3.9.7 version

As we will have control over these version, we can decide *when* to upgrade (if ever required) and we can roll back if the new pre-compiled version was built with a problem.

Note: Python versions may need to be re-compiled each time that the `-base` image is re-built. This is due that some underlying libraries that Python depend on may have changed.

Note: Installing always the latest version is harder to maintain. It will require building the newest version each time a new patch version is released. Beacause of that, Read the Docs will always be behind official releases. Besides, it will give projects different versions more often.

Exposing to the user the patch version would require to cache many different versions ourselves, and if the user selects one patched version that we don't have cached by mistake, those builds will add extra build time.

How do we add a Python version?

Adding a new Python version requires:

- pre-compile the desired version for each Ubuntu OS version supported
- upload the compressed output to S3
- add the supported version to the config file validator

How do we remove an old Python version?

At some point, an old version of Python will be deprecated (eg. 3.4) and will be removed. To achieve this, we can just remove the pre-compiled Python version from the cache.

However, unless it's strictly neeed for some specific reason, we shouldn't require to remove support for a Python version as long as we support the Ubuntu OS version where this version was compiled for.

In any case, we will know which projects are using these versions because they are pinning these specific versions in the config file. We could show a message in the build output page and also send them an email with the EOL date for this image.

However, removing pre-compiled Python version that it's being currently used by some users won't make their builds to fail. Instead, that Python version will be compiled and installed at build time; adding a "penalization" time to those projects and motivating them to move forward to a newer version.

How do we upgrade system versions?

We usually don't upgrade these dependencies unless we upgrade the Ubuntu version. So, they will be only upgraded when we go from Ubuntu 18.04 LTS to Ubuntu 20.04 LTS for example.

Examples of these versions are:

- doxygen
- git
- subversion
- pandoc
- swig
- latex

This case will introduce a new `base` image. Example, `ubuntu22-base` in 2022. Note that these images will be completely isolated from the rest and don't require them to rebuild. This also allow us to start testing a newer Ubuntu version (e.g. 22.04 LTS) without breaking people's builds, even before it's officially released.

How do we add an extra requirement?

In case we need to add an extra requirement to the `base` image, we will need to rebuild all of them. The new image *may have different package versions* since there may be updates on the Ubuntu repositories. This conveys some risk here, but in general we shouldn't require to add packages to the base images.

In case we need an extra requirement for *all our images*, I'd recommend to add it when creating a new base image.

If it's strongly needed and we can't wait for a new base image, we could install it at build time in a similar way as we do with `build.apt_packages` as a temporal workaround.

How do we create a mirror for each language?

A mirror can be created with `wget` together with `rclone`:

1. Download all the files from the official mirror:

```
# https://stackoverflow.com/questions/29802579/create-private-mirror-of-http-nodejs-
↪org-dist
wget --mirror --convert-links --adjust-extension --page-requisites --no-parent -e␣
↪robots=off http://nodejs.org/dist
```

2. Upload all the files to S3:

```
# https://rclone.org/s3/
rclone sync -i nodejs.org s3:languages
```

Note: Downloading a copy of the official mirror took 15m and 52Gb.

How local development will work with the new approach?

Local development will require scripts to clone the official mirrors for each language and upload them to MinIO (S3). Besides, a script to define a set of Python version, pre-compile them and also upload them to S3.

This is already covered by this simple bash script and tested in this PR with a POC: https://github.com/readthedocs/readthedocs.org/pull/8453

5.2.10 Deprecation plan

After this design document gets implemented and tested, all our current images (`stable`, `latest`, `testing`) will be deprecated and their usage will be de-motivated. However, we could keep them on our builders to give users a good time to migrate their projects to the new ones.

We may want to keep only the latest Ubuntu LTS release available in production, with a special consideration for our current Ubuntu 18.04 LTS on `stable`, `latest` and `testing` because 100% of the projects depend on them currently. Once Ubuntu 22.04 LTS is released, we should communicate that Ubuntu 20.04 LTS is deprecated, and keep it available in our servers during the time that's officially supported by Ubuntu during the "Maintenance updates" (see "Login term support and interim releases" in https://ubuntu.com/about/release-cycle). As an example, Ubuntu 22.04 LTS will be officially released on April 2022 and we will offer support for it until 2027.

Warning: Deleting `-base` images from the build servers **will make project's builds to fail**. We want to keep supporting them as much as we can, but having a well-defined deprecation policy is a win.

5.2.11 Work required and rollout plan

The following steps are required to support the full proposal of this document.

1. allow users to install extras languages requirements via config file
    - update config file to support `build.os` and `build.languages` config
    - modify builder code to run `asdf install` for all supported languages
2. build a new base Docker image with new structure (`ubuntu20-base`)
    - build new image with Ubuntu 20.04 LTS and pre-installed `asdf` with all its plugins
    - do not install any language version on base image
    - deploy builders with new base image

At this point, we will have a full working setup. It will be opt-in by using the new configs `build.os` and `build.languages`. However, *all languages* will be installed at build time; which will "penalize" all projects because all of them will have to install Python.

After testing this for some time, we can continue with the following steps that provides a cache to optimize installation times:

1. create mirrors on S3 for all supported languages
2. create mirror for pre-compiled latest 3 Python versions, Python 2.7 and PyPy3

5.2.12 Conclusion

There is no need to differentiate the images by its state (stable, latest, testing) but by its main base differences: OS. The version of the OS will change many library versions, LaTeX dependencies, basic required commands like git and more, that doesn't seem to be useful to have the same OS version with different states.

Allowing users to install extra languages by using the Config File will cover most of the support requests we have had in the past. It also will allow us to know more about how our users are using the platform to make future decisions based on this data. Exposing users how we want them to use our platform will allow us to be able to maintain it longer, than giving the option to select a specific Docker image by name that we can't guarrantee it will be frozen.

Finally, having the ability to deprecate and *remove* pre-built images from our builders over time, will reduce the maintainance work required from the the core team. We can always support all the languages versions by installing them at build time. The only required pre-built image for this are the OS `-base` images. In fact, even after decided to deprecate and removed a pre-built image from the builders, we can re-build it if we find that it's affecting many projects and slowing down their builds too much, causing us problems.

5.3 Embed APIv3

The Embed API allows users to embed content from documentation pages in other sites. It has been treated as an *experimental* feature without public documentation or real applications, but recently it started to be used widely (mainly because we created the `hoverxref` Sphinx extension).

The main goal of this document is to design a new version of the Embed API to be more user friendly, make it more stable over time, support embedding content from pages not hosted at Read the Docs, and remove some quirkiness that makes it hard to maintain and difficult to use.

Note: This work is part of the CZI grant that Read the Docs received.

5.3.1 Current implementation

The current implementation of the API is partially documented in Embedding Content From Your Documentation. It has some known problems:

- There are different ways of querying the API: `?url=` (generic) and `?doc=` (relies on Sphinx's specific concept)
- Doesn't support MkDocs
- Lookups are slow (~500 ms)
- IDs returned aren't well formed (like empty IDs `"headers": [{"title": "#"}]`)
- The content is always an array of one element
- It tries different variations of the original ID
- It doesn't return valid HTML for definition lists (`dd` tags without a `dt` tag)

5.3.2 Goals

We plan to add new features and define a contract that works the same for all HTML. This project has the following goals:

- Support embedding content from pages hosted outside Read the Docs
- Do not depend on Sphinx `.fjson` files
- Query and parse the `.html` file directly (from our storage or from an external request)
- Rewrite all links returned in the content to make them absolute
- Require a valid HTML `id` selector
- Accept only `?url=` request GET argument to query the endpoint
- Support `?nwords=` and `?nparagraphs=` to return chunked content
- Handle special cases for particular doctools (e.g. Sphinx requires to return the `.parent()` element for `dl`)
- Make explicit the client is asking to handle the special cases (e.g. send `?doctool=sphinx&version=4.0.1&writer=html4`)
- Delete HTML tags from the original document (for well-defined special cases)

- Add HTTP cache headers to cache responses
- Allow CORS from everywhere *only* for public projects

5.3.3 The contract

Return the HTML tag (and its children) with the `id` selector requested and replace all the relative links from its content making them absolute.

Note: Any other case outside this contract will be considered *special* and will be implemented only under `?doctool=`, `?version=` and `?writer=` arguments.

If no `id` selector is sent to the request, the content of the first meaningful HTML tag (`<main>`, `<div role="main">` or other well-defined standard tags) identifier found is returned.

5.3.4 Embed endpoints

This is the list of endpoints to be implemented in APIv3:

GET /api/v3/embed/

Returns the exact HTML content for a specific identifier (`id`). If no anchor identifier is specified the content of the first one returned.

Example request:

```
$ curl https://readthedocs.org/api/v3/embed/?url=https://docs.readthedocs.io/en/
↪latest/development/install.html#set-up-your-environment
```

Example response:

```
{
   "project": "docs",
   "version": "latest",
   "language": "en",
   "path": "development/install.html",
   "title": "Development Installation",
   "url": "https://docs.readthedocs.io/en/latest/install.html#set-up-your-
↪environment",
   "id": "set-up-your-environment",
   "content": "<div class=\"section\" id=\"development-installation\">\n<h1>
↪Development Installation<a class=\"headerlink\" href=\"https://docs.readthedocs.
↪io/en/stable/development/install.html#development-installation\" title=\
↪"Permalink to this headline\">¶</a></h1>\n ..."
}
```

Query Parameters

- **(required)** (*url*) – Full URL for the documentation page with optional anchor identifier.

GET /api/v3/embed/metadata/

Returns all the available metadata for an specific page.

Note: As it's not trivial to get the `title` associated with a particular `id` and it's not easy to get a nested list of identifiers, we may not implement this endpoint in initial version.

The endpoint as-is, is mainly useful to explore/discover what are the identifiers available for a particular page –which is handy in the development process of a new tool that consumes the API. Because of this, we don't have too much traction to add it in the initial version.

Example request:

```
$ curl https://readthedocs.org/api/v3/embed/metadata/?url=https://docs.readthedocs.
↪io/en/latest/development/install.html
```

Example response:

```
{
  "identifiers": {
      "id": "set-up-your-environment",
      "url": "https://docs.readthedocs.io/en/latest/development/install.html#set-up-
↪your-environment"
      "_links": {
          "embed": "https://docs.readthedocs.io/_/api/v3/embed/?url=https://docs.
↪readthedocs.io/en/latest/development/install.html#set-up-your-environment"
      }
  },
  {
      "id": "check-that-everything-works",
      "url": "https://docs.readthedocs.io/en/latest/development/install.html#check-
↪that-everything-works"
      "_links": {
          "embed": "https://docs.readthedocs.io/_/api/v3/embed/?url=https://docs.
↪readthedocs.io/en/latest/development/install.html#check-that-everything-works"
      }
   },
}
```

Query Parameters

- **(required)** (*url*) – Full URL for the documentation page

5.3.5 Handle specific Sphinx cases

We are currently handling some special cases for Sphinx due how it writes the HTML output structure. In some cases, we look for the HTML tag with the identifier requested but we return the `.next()` HTML tag or the `.parent()` tag instead of the *requested one*.

Currently, we have identified that this happens for definition tags (`dl`, `dt`, `dd`) –but may be other cases we don't know yet. Sphinx adds the `id=` attribute to the `dt` tag, which contains only the title of the definition, but as a user, we are expecting the description of it.

In the following example we will return the whole `dl` HTML tag instead of the HTML tag with the identifier `id="term-name"` as requested by the client, because otherwise the "Term definition for Term Name" content won't be included and the response would be useless.

```html
<dl class="glossary docutils">
  <dt id="term-name">Term Name</dt>
  <dd>Term definition for Term Name</dd>
</dl>
```

If the definition list (`dl`) has more than *one definition* it will return **only the term requested**. Considering the following example, with the request `?url=glossary.html#term-name`

```html
<dl class="glossary docutils">
  ...

  <dt id="term-name">Term Name</dt>
  <dd>Term definition for Term Name</dd>

  <dt id="term-unknown">Term Unknown</dt>
  <dd>Term definition for Term Unknown </dd>

  ...
</dl>
```

It will return the whole `dl` with only the `dt` and `dd` for `id` requested:

```html
<dl class="glossary docutils">
  <dt id="term-name">Term Name</dt>
  <dd>Term definition for Term Name</dd>
</dl>
```

However, this assumptions may not apply to documentation pages built with a different doctool than Sphinx. For this reason, we need to communicate to the API that we want to handle this special cases in the backend. This will be done by appending a request GET argument to the Embed API endpoint: `?doctool=sphinx&version=4.0.1&writer=html4`. In this case, the backend will known that has to deal with these special cases.

Note: This leaves the door open to be able to support more special cases (e.g. for other doctools) without breaking the actual behavior.

5.3.6 Support for external documents

When the `?url=` argument passed belongs to a documentation page not hosted on Read the Docs, the endpoint will do an external request to download the HTML file, parse it and return the content for the identifier requested.

The whole logic should be the same, the only difference would be where the source HTML comes from.

Warning: We should be careful with the URL received from the user because those may be internal URLs and we could be leaking some data. Example: `?url=http://localhost/some-weird-endpoint` or `?url=http://169.254.169.254/latest/meta-data/` (see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instancedata-data-retrieval.html).

This is related to SSRF (https://en.wikipedia.org/wiki/Server-side_request_forgery). It doesn't seem to be a huge problem, but something to consider.

Also, the endpoint may need to limit the requests per-external domain to avoid using our servers to take down another site.

Note: Due to the potential security issues mentioned, we will start with an allowed list of domains for common Sphinx docs projects. Projects like Django and Python, where `sphinx-hoverxref` users might commonly want to embed from. We aren't planning to allow arbitrary HTML from any website.

5.3.7 Handle project's domain changes

The proposed Embed APIv3 implementation only allows `?url=` argument to embed content from that page. That URL can be:

- a URL for a project hosted under `<project-slug>.readthedocs.io`
- a URL for a project with a custom domain

In the first case, we can easily get the project's slug directly from the URL. However, in the second case we get the project's slug by querying our database for a `Domain` object with the full domain from the URL.

Now, consider that all the links in the documentation page that uses Embed APIv3 are pointing to `docs.example.com` and the author decides to change the domain to be `docs.newdomain.com`. At this point there are different possible scenarios:

- The user creates a new `Domain` object with `docs.newdomain.com` as domain's name. In this case, old links will keep working because we still have the old `Domain` object in our database and we can use it to get the project's slug.
- The user *deletes* the old `Domain` besides creating the new one. In this scenario, our query for a `Domain` with name `docs.example.com` to our database will fail. We will need to do a request to `docs.example.com` and check for a 3xx response status code and in that case, we can read the `Location:` HTTP header to find the new domain's name for the documentation. Once we have the new domain from the redirect response, we can query our database again to find out the project's slug.

  Note: We will follow up to 5 redirects to find out the project's domain.

5.3.8 Embed APIv2 deprecation

The v2 is currently widely used by projects using the `sphinx-hoverxref` extension. Because of that, we need to keep supporting it as-is for a long time.

Next steps on this direction should be:

- Add a note in the documentation mentioning this endpoint is deprecated
- Promote the usage of the new Embed APIv3
- Migrate the `sphinx-hoverxref` extension to use the new endpoint

Once we have done them, we could check our NGINX logs to find out if there are people still using APIv2, contact them and let them know that they have some months to migrate since the endpoint is deprecated and will be removed.

5.3.9 Unanswered questions

- How do we distinguish between our APIv3 for resources (models in the database) from these "feature API endpoints"?

5.4 In Doc Search UI

Giving readers the ability to easily search the information that they are looking for is important for us. We have already upgraded to the latest version of Elasticsearch and we plan to implement `search as you type` feature for all the documentations hosted by us. It will be designed to provide instant results as soon as the user starts typing in the search bar with a clean and minimal frontend. This design document aims to provides the details of it. This is a GSoC'19 project.

Warning: This design document details future features that are **not yet implemented**. To discuss this document, please get in touch in the issue tracker.

The final result may look something like this:

Fig. 1: Short demo

5.4.1 Goals And Non-Goals

Project Goals

- Support a search-as-you-type/autocomplete interface.
- Support across all (or virtually all) Sphinx themes.
- Support for the JavaScript user experience down to IE11 or graceful degradation where we can't support it.
- Project maintainers should have a way to opt-in/opt-out of this feature.
- (Optional) Project maintainers should have the flexibility to change some of the styles using custom CSS and JS files.

Non-Goals

- For the initial release, we are targeting only Sphinx documentations as we don't index MkDocs documentations to our Elasticsearch index.

5.4.2 Existing Search Implementation

We have a detailed documentation explaining the underlying architecture of our search backend and how we index documents to our Elasticsearch index. You can read about it *here*.

5.4.3 Proposed Architecture for In-Doc Search UI

Frontend

Technologies

Frontend is to designed in a theme agnostics way. For that, we explored various libraries which may be of use but none of them fits our needs. So, we might be using vanilla JavaScript for this purpose. This will provide us some advantages over using any third party library:

- Better control over the DOM.
- Performance benefits.

Proposed Architecture

We plan to select the search bar, which is present in every theme, and use the querySelector() method of JavaScript. Then add an event listener to it to listen for the changes and fire a search query to our backend as soon as there is any change. Our backend will then return the suggestions, which will be shown to the user in a clean and minimal UI. We will be using document.createElement() and node.removeChild() method provided by JavaScript as we don't want empty `<div>` hanging out in the DOM.

We have a few ways to include the required JavaScript and CSS files in all the projects:

- Add CSS into `readthedocs-doc-embed.css` and JS into `readthedocs-doc-embed.js` and it will get included.
- Package the in-doc search into it's own self-contained CSS and JS files and include them in a similar manner to `readthedocs-doc-embed.*`.
- It might be possible to package up the in-doc CSS/JS as a sphinx extension. This might be nice because then it's easy to enable it on a per-project basis. When we are ready to roll it out to a wider audience, we can make a decision to just turn it on for everybody (put it in here) or we could enable it as an opt-in feature like the 404 extension.

UI/UX

We have two ways which can be used to show suggestions to the user.

- Show suggestions below the search bar.
- Open a full page search interface when the user click on search field.

Backend

We have a few options to support `search as you type` feature, but we need to decide that which option would be best for our use-case.

Edge NGram Tokenizer

- Pros
  - More effective than Completion Suggester when it comes to autocompleting words that can appear in any order.
  - It is considerable fast because most of the work is being done at index time, hence the time taken for autocompletion is reduced.
  - Supports highlighting of the matching terms.
- Cons
  - Requires greater disk space.

Completion Suggester

- Pros
  - Really fast as it is optimized for speed.
  - Does not require large disk space.
- Cons
  - Matching always starts at the beginning of the text. So, for example, "Hel" will match "Hello, World" but not "World Hello".
  - Highlighting of the matching words is not supported.
  - According to the official docs for Completion Suggester, fast lookups are costly to build and are stored in-memory.

5.4.4 Milestones

Milestone	Due Date
A local implementation of the project.	12th June, 2019
In-doc search on a test project hosted on Read the Docs using the RTD Search API.	20th June, 2019
In-doc search on docs.readthedocs.io.	20th June, 2019
Friendly user trial where users can add this on their own docs.	5th July, 2019
Additional UX testing on the top-10 Sphinx themes.	15th July, 2019
Finalize the UI.	25th July, 2019
Improve the search backend for efficient and fast search results.	10th August, 2019

Open Questions

- Should we rely on jQuery, any third party library or pure vanilla JavaScript?
- Are the subprojects to be searched?
- Is our existing Search API is sufficient?
- Should we go for edge ngrams or completion suggester?

5.5 Proposed contents for new Sphinx guides

Note: This work is in progress, see discussion on this Sphinx issue and the pull requests linked at the end.

The two main objectives are:

- Contributing a good Sphinx tutorial for beginners. This should introduce the readers to all the various Sphinx major features in a pedagogical way, and be mostly focused on Markdown using MyST. We would try to find a place for it in the official Sphinx documentation.
- Write a new narrative tutorial for Read the Docs that complements the existing guides and offers a cohesive story of how to use the service.

5.5.1 Sphinx tutorial

Appendixes are optional, i.e. not required to follow the tutorial, but highly recommended.

1. The Sphinx way
   - Preliminary section giving an overview of what Sphinx is, how it works, how reStructuredText and Markdown/MyST are related to it, some terminology (toctree, builders), what can be done with it.
2. About this tutorial
   - A section explaining the approach of the tutorial, as well as how to download the result of each section for closer inspection or for skipping parts of it.
3. Getting started
   1. Creating our project
      - Present a fictitious goal for a documentation project
      - Create a blank `README.md` to introduce the most basic elements of Markdown (headings and paragraph text)
   2. Installing Sphinx and cookiecutter in a new development environment
      - Install Python (or miniforge)
      - Create a virtual environment (and/or conda environment)
      - Activate our virtual environment (it will always be the first step)
      - Install Sphinx inside the virtual environment
      - Check that `sphinx-build --help` works (yay!)
   3. Creating the documentation layout

      - Apply our cookiecutter to create a minimal `docs/` directory (similar to what `sphinx-quickstart` does, but with source and build separation by default, project release 0.1, English language, and a MyST index, if at all)[1]
      - Check that the correct files are created (yay!)
   4. Appendix: Using version control
      - Install git (we will not use it during the tutorial)
      - Add a proper `.gitignore` file (copied from gitignore.io)
      - Create the first commit for the project (yay!)
4. First steps to document our project using Sphinx
   1. Converting our documentation to local HTML
      - Create (or minimally tweak) `index.md`
      - Build the HTML output using `sphinx-build -b -W html doc doc/_build/html`[2]
      - Navigate to `doc/_build/html` and launch an HTTP server (`python -m http.server`)
      - Open http://localhost:8000 in a web browser, and see the HTML documentation (yay!)
   2. Converting our documentation to other formats
      - Build PseudoXML using `make pseudoxml`
      - Build Text using `make text`
      - See how the various formats change the output (yay!)
   3. Appendix: Simplify documentation building by using Make[3]
      - Install Make (nothing is needed on Windows, `make.bat` is standalone)
      - Add more content to `index.md`
      - Build HTML doing `cd doc && make html`
      - Observe that the HTML docs have changed (yay!)
   4. Appendix: PDF without LaTeX using rinoh (beta)
5. Customizing Sphinx configuration
   1. Changing the HTML theme
      - Install https://pypi.org/project/furo/
      - Change the `html_theme` in `conf.py`
      - Rebuild the HTML documentation and observe that the theme has changed (yay!)
   2. Changing the PDF appearance
      - Add a `latex_theme` and set it to `howto`
      - Rebuild `make latexpdf`
      - Check that the appearance changed (yay!)
   3. Enable an extension

[1] Similar to https://github.com/sphinx-contrib/cookiecutter, but only for the `docs/` directory? This way it can be less opinionated about everything else

[2] At first I considered "make mode", but the current maintainers don't know much about its original intent (see my comment here and the discussion after it)

[3] There have been attempts at creating a `sphinx` command, see this pull request

    - Add a string to the `extensions` list in `conf.py` for `sphinx.ext.duration`
    - Rebuild the HTML docs `make html` and notice that now the times are printed (yay!)

6. Writing narrative documentation with Sphinx
    - First focus on `index.md`, diving more into Markdown and mentioning Semantic Line Breaks.
    - Then add another `.md` file to teach how `toctree` works.
    - Then continue introducing elements of the syntax to add pictures, cross-references, and the like.
7. Describing code in Sphinx
    - Explain the Python domain as part of narrative documentation to interleave code with text, include doctests, and justify the usefulness of the next section.
8. Autogenerating documentation from code in Sphinx
9. Deploying a Sphinx project online
    - A bit of background on the options: GitHub/GitLab Pages, custom server, Netlify, Read the Docs
    - Make reference to Read the Docs tutorial
10. Appendix: Using Jupyter notebooks inside Sphinx
11. Appendix: Understanding the docutils document tree
12. Appendix: Where to go from here
    - Refer the user to the Sphinx, reST and MyST references, prominent projects already using Sphinx, compilations of themes and extensions, the development documentation.

5.5.2 Read the Docs tutorial

1. The Read the Docs way
2. Getting started
    1. Preparing our project on GitHub
        - Fork a starter GitHub repository (something like our demo template, as a starting point that helps mimicking the `sphinx-quickstart` or `cookiecutter` step without having to checkout the code locally)
    2. Importing our project to Read the Docs
        - Sign up with GitHub on RTD
        - Import the project (don't "Edit advanced project options", we will do this later)
        - The project is created on RTD
        - Browse "builds", open the build live logs, wait a couple of minutes, open the docs (yay!)
    3. Basic configuration changes
        - Add a description, homepage, and tags
        - Configure your email for build failure notification (until we turn them on by default)
        - Enable "build pull requests for this project" in the advanced settings
        - Edit a file from the GitHub UI as part of a new branch, and open a pull request
        - See the RTD check on the GitHub PR UI, wait a few minutes, open result (yay!)

3. Customizing the build process
   - Use `readthedocs.yaml` (rather than the web UI) to customize build formats, change build requirements and Python version, enable fail-on-warnings
4. Versioning documentation
   - Explain how to manage versions on RTD: create release branches, activate the corresponding version, browse them in the version selector, selectively build versions
   - Intermediate topics: hide versions, create Automation Rules
5. Getting insights from your projects
   - Move around the project, explore results in Traffic Analytics
   - Play around with server-side search, explore results in Search Analytics
6. Managing translations
7. Where to go from here
   - Reference our existing guides, prominent projects already using RTD, domain configuration, our support form, our contributing documentation

5.5.3 Possible new how-to Guides

Some ideas for extra guides on specific topics, still for beginners but more problem-oriented documents, covering a wide range of use cases:

- How to turn a bunch of Markdown files into a Sphinx project
- How to turn a bunch of Jupyter notebooks into a Sphinx project
- How to localize an existing Sphinx project
- How to customize the appearance of the HTML output of a Sphinx project
- How to convert existing reStructuredText documentation to Markdown
- How to use Doxygen autogenerated documentation inside a Sphinx project
- How to keep a changelog of your project

5.5.4 Reference

All the references should be external: the Sphinx reference, the MyST and reST syntax specs, and so forth.

5.6 Organizations

Currently we don't support organizations in the community site (a way to group different projects), we only support individual accounts.

Several integrations that we support like GitHub and Bitbucket have organizations, where users group their repositories and manage them in groups rather than individually.

5.6.1 Why move organizations in the community site?

We support organizations in the commercial site, having no organizations in the community site makes the code maintenance difficult for Read the Docs developers. Having organizations in the community site will make the differences between both more easy to manage.

Users from the community site can have organizations in external sites from where we import their projects (like GitHub, Gitlab). Currently users have all projects from different organizations in their account. Having not a clear way to group/separate those.

We are going to first move the code, and after that enable the feature on the community site.

5.6.2 How are we going to support organizations?

Currently only users can own projects in the community site. With organizations this is going to change to: Users and organizations can own projects.

With this, the migration process would be straightforward for the community site.

For the commercial site we are only to allow organizations to own projects for now (since the we have only subscriptions per organizations).

5.6.3 What features of organizations are we going to support?

We have the following features in the commercial site that we don't have on the community site:

- Owners
- Teams
- Permissions
- Subscriptions

Owners should be included to represent owners of the current organization.

Teams, this is also handy to manage access to different projects under the same organization.

Permissions, currently we have two type of permissions for teams: admin and read only. Read only permissions doesn't make sense in the community site since we only support public projects/versions (we do support private versions now, but we are planning to remove those). So, we should only support admin permissions for teams.

Subscriptions, this is only valid for the corporate site, since we don't charge for use in the community site.

5.6.4 How to migrate current projects

Since we are not replacing the current implementation, we don't need to migrate current projects from the community site nor from the corporate site.

5.6.5 How to migrate the organizations app

The migration can be split in:

1. Remove/simplify code from the organizations app on the corporate site.
2. Isolate/separate models and code that isn't going to be moved.
3. Start by moving the models, managers, and figure out how to handle migrations.
4. Move the rest of the code as needed.
5. Activate organizations app on the community site.
6. Integrate the code from the community site to the new code.
7. UI changes

We should start by removing unused features and dead code from the organizations in the corporate site, and simplify existing code if possible (some of this was already done).

Isolate/separate the models to be moved from the ones that aren't going to be moved. We should move the models that aren't going to me moved to another app.

- Plan
- PlanFeature
- Subscription

This app can be named *subscriptions*. We can get around the table names and migrations by setting the explicitly the table name to `organizations_<model>`, and doing a fake migration. Following suggestions in https://stackoverflow.com/questions/48860227/moving-multiple-models-from-one-django-app-to-another, that way we avoid having any downtime during the migration and any inconvenient caused from renaming the tables manually.

Code related to subscriptions should be moved out from the organizations app.

After that, it should be easier to move the organizations *app* (or part of it) to the community site (and no changes to table names would be required).

We start by moving the models.

- Organization
- OrganizationOwner
- Team
- TeamInvite
- TeamMember

Migrations aren't moved, since all current migrations depend on other models that aren't going to be moved. In the community site we run an initial migration, for the corporate site we run a fake migration. The migrations left from the commercial site can be removed after that.

For managers and querysets that depend on subscriptions, we can use our pattern to make overridable classes (inheriting from `SettingsOverrideObject`).

Templates, urls, views, forms, notifications, signals, tasks can be moved later (we just need to make use of the models from the `readthedocs.organizations` module).

If we decide to integrate organizations in the community site, we can add/move the UI elements and enable the app.

After the app is moved, we can move more code that depends on organizations to the community site.

5.6.6 Namespace

Currently we use the project's slug as namespace, in the commercial site we use the combination of `organization.slug` + `project.slug` as namespace, since in the corporate site we don't care so much about a unique namespace between all users, but a unique namespace per organization.

For the community site probably this approach isn't the best, since we always serve docs publicly from `slug.readthedocs.io`. And most of the users don't have a custom domain.

The corporate site will use `organization.slug` + `project.slug` as slug, And the community site will always use `project.slug` as slug, even if the project belongs to an organization.

We need to refactor the way we get the namespace to be more easy to manage in both sites.

5.6.7 Future Changes

Changes that aren't needed immediately after the migration, but that should be done:

- UI for organizations in the community site.
- Add new endpoints to the API (v3 only).
- Make the relationship between the models `Organization` and `Project` one to many (currently many to many).

5.7 Design of Pull Request Builder

5.7.1 Background

This will focus on automatically building documentation for Pull Requests on Read the Docs projects. This is one of the most requested feature of Read the Docs. This document will serve as a design document for discussing how to implement this features.

5.7.2 Scope

- Making Pull Requests work like temporary `Version`
- Excluding PR Versions from Elasticsearch Indexing
- Adding a `PR Builds` Tab in the Project Dashboard
- Updating the Footer API
- Adding Warning Banner to Docs
- Serving PR Docs
- Excluding PR Versions from Search Engines
- Receiving `pull_request` webhook event from Github
- Fetching data from pull requests
- Storing PR Version build Data
- Creating PR Versions when a pull request is opened and Triggering a build
- Triggering Builds on new commits on a PR
- Status reporting to Github

5.7.3 Fetching Data from Pull Requests

We already get Pull request events from Github webhooks. We can utilize that to fetch data from pull requests. when a `pull_request` event is triggered we can fetch the data of that pull request. We can fetch the pull request by doing something similar to travis-ci. ie: `git fetch origin +refs/pull/<pr_number>/merge:`

5.7.4 Modeling Pull Requests as a Type of Version

Pull requests can be Treated as a Type of Temporary `Version`. We might consider adding a `VERSION_TYPES` to the `Version` model.

- If we go with `VERSION_TYPES` we can add something like `pull_request` alongside Tag and Branch.

We should add `Version` and `Build` Model Managers for PR and Regular Versions and Builds. The proposed names for PR and Regular Version and Build Mangers are `external` and `internal`.

We can then use `Version.internal.all()` to get all regular versions, `Version.external.all()` to get all PR versions.

We can then use `Build.internal.all()` to get all regular version builds, `Build.external.all()` to get all PR version builds.

5.7.5 Excluding PR Versions from Elasticsearch Indexing

We should exclude to PR Versions from being Indexed to Elasticsearch. We need to update the queryset to exclude PR Versions.

5.7.6 Adding a PR Builds Tab in the Project Dashboard

We can add a Tab in the project dashboard that will listout the PR Builds of that project. We can name it `PR Builds`.

5.7.7 Creating Versions for Pull Requests

If the Github webhook event is `pull_request` and action is `opened`, this means a pull request was opened in the projects repository. We can create a `Version` from the Payload data and trigger a initial build for the version. A version will be created whenever RTD receives an event like this.

5.7.8 Triggering Build for New Commits in a Pull Request

We might want to trigger a new build for the PR version if there is a new commit on the PR. If the Github webhook event is `pull_request` and action is `synchronize`, this means a new commit was added to the pull request.

5.7.9 Status Reporting to Github

We could send build status reports to Github. We could send if the build was Successful or Failed. We can also send the build URL. By this we could show if the build passed or failed on Github something like travis-ci does.

As we already have the `repo:status` scope on our OAuth App, we can send the status report to Github using the Github Status API.

Sending the status report would be something like this:

`POST /repos/:owner/:repo/statuses/:sha`

```
{
    "state": "success",
    "target_url": "<pr_build_url>",
    "description": "The build succeeded!",
    "context": "continuous-documentation/read-the-docs"
}
```

5.7.10 Storing Pull Request Docs

We need to think about how and where to store data after a PR Version build is finished. We can store the data in a blob storage.

5.7.11 Excluding PR Versions from Search Engines

We should Exclude the PR Versions from Search Engines, because it might cause problems for RTD users. As users might land to a pull request doc but not the original Project Docs. This will cause confusion for the users.

5.7.12 Serving PR Docs

We need to think about how we want to serve the PR Docs.

- We could serve the PR Docs from another Domain.
- We could serve the PR Docs using `<pr_number>` namespace on the same Domain.
  - Using `pr-<pr_number>` as the version slug `https://<project_slug>.readthedocs.io/<language_code>/pr-<pr_number>/`
  - Using `pr` subdomain `https://pr.<project_slug>.readthedocs.io/<pr_number>/`

5.7.13 Updating the Footer API

We need to update the Footer API to reflect the changes. We might want to have a way to show that if this is a PR Build on the Footer.

- For regular project docs we should remove the PR Versions from the version list of the Footer.
- We might want to send `is_pr` data with the Footer API response.

5.7.14 Adding Warning Banner to Docs

We need to add a warning banner to the PR Version Docs to let the users know that this is a Draft/PR version. We can use a sphinx extension that we will force to install on the PR Versions to add the warning banner.

5.7.15 Related Issues

- Autobuild Docs for Pull Requests
- Add travis-ci style pull request builder

5.8 Privacy Levels

This document describes how to handle and unify privacy levels on the community and commercial version of Read the Docs.

5.8.1 Current state

Currently, we have three privacy levels for projects and versions:

1. Public
2. Private
3. Protected (currently hidden)

These levels of privacy aren't clear and bring confusion to our users. Also, the private level doesn't makes sense on the community site, since we only support public projects.

Places where we use the privacy levels are:

- On serving docs
- Footer
- Dashboard

5.8.2 Project level privacy

Project level privacy was meant to control the dashboard visibility.

This privacy level brings to confusion when users want to make a version public. We should remove all the project privacy levels.

For the community site the dashboard would be always visible, and for the commercial site, the dashboard would be always hidden.

The project privacy level is also used to serve the `404.html` page, show `robots.txt`, and show `sitemap.xml`. The privacy level from versions should be used instead.

Some other ideas about keeping the privacy level is to dictate the default version level of new versions, but removing all other logic related to this privacy level. This can be (or is going to be) possible with automation rules, so we can just remove the field.

5.8.3 Version level privacy

Version level privacy is mainly used to restrict access to documentation. For public level, everyone can access to the documentation. For private level, only users that are maintainers or that belong to a team with access (for the commercial site) can access to the documentation.

The protected privacy level was meant to hide versions from listings and search. For the community site these versions are treated like public versions, and on the commercial site they are treated like private.

The protected privacy level is currently hidden. To keep the behavior of hiding versions from listings and search, a new field should be added to the Version model and forms: `hidden` (#5321). The privacy level (public or private) would be respected to determine access to the documentation.

For the community site, the privacy level would be public and can't be changed.

The default privacy level of new versions for the commercial site would be `private` (this is the `DEFAULT_PRIVACY_LEVEL` setting).

5.8.4 Footer

The footer is used to display not hidden versions that the current user has access to.

For the community site no changes are required on the footer.

For the commercial site we use the project level privacy to decide if show or not links to the project's dashboard: downloads, project home, and builds. Given that the project privacy level would be removed (and the dashboard is always under login), those links would never be shown, except for admin users (owners or from a team with admin access) since they are the only ones allowed to make changes on the project.

5.8.5 Overview

For the community site:

- The project's dashboard is visible to all users.
- All versions are always public.
- The footer shows links to the project's dashboard (build, downloads, home) to all users.
- Only versions with `hidden = False` are listed on the footer and appear on search results.
- If a project has a `404.html` file on the default version, it's served.
- If a project has a `robots.txt` file on the default version, it's served.
- A `sitemap.xml` file is always served.

For the commercial site:

- The project's dashboard is visible to only users that have read permission over the project.
- The footer shows links to the project's dashboard (build, downloads, home) to only admin users.
- Only versions with `hidden = False` are listed on the footer and appear on search results.
- If a project has a `404.html` file on the default version, it's served if the user has permission over that version.
- If a project has a `robots.txt` file on the default version, it's served if the user has permission over that version.
- A `sitemap.xml` file is served if the user has at least one public version. And it will only list public versions.

5.8.6 Migration

To differentiate between allowing or not privacy levels, we need to add a setting `RTD_ALLOW_PRIVACY_LEVELS` (`False` by default).

For the community and commercial site, we need to:

- Remove/change code that depends on the project's privacy level. Use the global setting `RTD_ALLOW_PRIVACY_LEVELS` and default version's privacy level instead.
  - Display robots.txt
  - Serve 404.html page
  - Display sitemap.xml
  - Querysets
- Remove `Project.privacy_level` field
- Migrate all protected versions to have the attribute `hidden = True` (data migration), and set their privacy level to public for the community site and private for the commercial site.
- Change all querysets used to list versions on the footer and on search to use the `hidden` attribute.
- Update docs

For the community site:

- Hide all privacy level related settings from the version form.
- Don't expose privacy levels on API v3.
- Mark all versions as public.

For the commercial site:

- Always hide the dashboard
- Show links to the dashboard (downloads, builds, project home) on the footer only to admin users.

5.8.7 Upgrade path overview

Community site

The default privacy level for the community site is public for versions and the dashboard is always public.

Public project (community)

- Public version: Normal use case, no changes required.
- Protected version: Users didn't want to list this version on the footer, but also not deactivate it. We can do a data migration of those versions to the new `hidden` setting and make them public.
- Private version: Users didn't want to show this version to their users yet or they were testing something. This can be solved with the pull request builder feature and the `hidden` setting. We migrate those to public with the `hidden` setting. If we are worried about leaking anything from the version, we can email users before doing the change.

Protected project (community)

Protected projects are not listed publicly. Probably users were hosting a WIP project, or personal public project. A public project should work for them, as we are removing listing all projects publicly (except for search).

The migration path for versions of protected projects is the same as a public project.

Private project (community)

Probably these users want to use our enterprise solution instead. Or they were hosting a personal project.

The migration path for versions of private projects is the same as a public project.

If we are worried about leaking anything from the dashboard or build page, we can email users before doing the change.

Commercial site

The default privacy level for the commercial site is private for versions and the dashboard is show only to admin users.

Private project (commercial)

- Private version: Normal usa case, not changes required.
- Protected version: Users didn't want to list this version on the footer, but also not deactivate it. This can be solved by using the new `hidden` setting. We can do a data migration of those versions to the new `hidden` setting and make them private.
- Public version: User has private code, but want to make public their docs. No changes required.

Protected project (commercial)

I can't think of a use case for protected projects, since they aren't listed publicly on the commercial site.

The migration path for versions of protected projects is the same as a private project.

Public project (commercial)

Currently we show links back to project dashboard if the project is public, which probably users shouldn't see. With the implementation of this design doc, public versions don't have links to the project dashboard (except for admin users) and the dashboard is always under login.

- Private versions: Users under the organization can see links to the dashboard. Not changes required.
- Protected versions: Users under the organization can see links to the dashboard. We can do a data migration of those versions to the new `hidden` setting and make them private.
- Public versions: All users can see links to the dashboard. Probably they have an open source project, but they still want to manage access using the same teams of the organization. Not changes are required.

A breaking change here is: users outside the organization would not be able to see the dashboard of the project.

5.9 Refactor `RemoteRepository` object

This document describes the current usage of `RemoteRepository` objects and proposes a new normalized modeling.

5.9.1 Goals

- De-duplicate data stored in our database.
- Save only one `RemoteRepository` per GitHub repository.
- Use an intermediate table between `RemoteRepository` and `User` to store associated remote data for the specific user.
- Make this model usable from our SSO implementation (adding `remote_id` field in `Remote` objects).
- Use Post `JSONField` to store associated `json` remote data.
- Make `Project` connect directly to `RemoteRepository` without being linked to a specific `User`.
- Do not disconnect `Project` and `RemoteRepository` when a user delete/disconnects their account.

5.9.2 Non-Goals

- Keep `RemoteRepository` in sync with GitHub repositories.
- Delete `RemoteRepository` objects deleted from GitHub.
- Listen to GitHub events to detect `full_name` changes and update our objects.

Note: We may need/want some of these non-goals in the future. They are just outside the scope of this document.

5.9.3 Current Implementation

When a user connect their account to a social account, we create a

- `allauth.socialaccount.models.SocialAccount` * basic information (provider, last login, etc) * provider's specific data saved in a JSON under `extra_data`
- `allauthsocialaccount.models.SocialToken` * token to hit the API on behalf the user

We *don't create* any `RemoteRepository` at this point. They are created when the user jumps into "Import Project" page and hit the circled arrows. It triggers `sync_remote_repostories` task in background that updates or creates `RemoteRepositories`, but **it does not delete** them (after #7183 and #7310 got merged, they will be deleted). One `RemoteRepository` is created per repository the `User` has access to.

Note: In corporate, we are automatically syncing `RemoteRepository` and `RemoteOganization` at signup (foreground) and login (background) via a signal. We should eventually move these to community.

5.9.4 Where `RemoteRepository` is used?

- List of available repositories to import under "Import Project"
- Show a "+", "External Arrow" or a "Lock" sign next to the element in the list * +: it's available to be imported * External Arrow: the repository is already imported (see RemoteRepository.matches method) * Lock: user doesn't have (admin) permissions to import this repository (uses `RemoteRepository.private` and `RemoteRepository.admin`)
- Avatar URL in the list of project available to import
- Update webhook when user clicks "Resync webhook" from the *Admin* > *Integrations* tab
- Send build status when building Pull Requests

5.9.5 New Normalized Implementation

The `ManyToMany` relation `RemoteRepository.users` will be changed to be `ManyToMany(through='RemoteRelation')` to add extra fields in the relation that are specific only for the User. Allows us to have *only one* `RemoteRepository` per GitHub repository with multiple relationships to `User`.

With this modeling, we can avoid the disconnection `Project` and `RemoteRepository` only by removing the `RemoteRelation`.

Note: All the points mentioned in the previous section may need to be adapted to use the new normalized modeling. However, it may be only field renaming or small query changes over new fields.

Use this modeling for SSO

We can get the list of `Project` where a user as access:

```
admin_remote_repositories = RemoteRepository.objects.filter(
    users__contains=request.user,
    users__remoterelation__admin=True,  # False for read-only access
)
Project.objects.filter(remote_repository__in=admin_remote_repositories)
```

5.9.6 Rollout Plan

Due the constraints we have in the `RemoteRepository` table and its size, we can't just do the data migration at the same time of the deploy. Because of this we need to be more creative here and find a way to re-sync the data from VCS providers, while the site continue working.

To achieve this, we thought on following this steps:

1. modify all the Python code to use the new modeling in .org and .com (will help us to find out bugs locally in an easier way) 1. QA this locally with test data 1. enable Django signal to re-sync RemoteRepository on login async (we already have this in .com). New active users will have updated data immediately 1. spin up a new instance with the new refactored code 1. run migrations to create a new table for `RemoteRepository` 1. re-sync everything from VCS providers into the new table for 1-week or so 1. dump-n-load `Project - RemoteRepository` relations 1. create a migration to use the new table with synced data 1. deploy new code once the sync is finished

See these issues for more context: * https://github.com/readthedocs/readthedocs.org/pull/7536#issuecomment-724102640 * https://github.com/readthedocs/readthedocs.org/pull/7675#issuecomment-732756118

5.10 Allow Installation of System Packages

Currently we don't allow executing arbitrary commands in the build process. The more common use case is to install extra dependencies.

5.10.1 Current status

There is a workaround when using Sphinx to run arbitrary commands, this is executing the commands inside the `conf.py` file. There isn't a workaround for MkDocs, but this problem is more common in Sphinx, since users need to install some extra dependencies in order to use autodoc or build Jupyter Notebooks.

However, installation of some dependencies require root access, or are easier to install using `apt`. Most of the CI services allow to use `apt` or execute any command with `sudo`, so users are more familiar with that workflow.

Some users use Conda instead of pip to install dependencies in order to avoid these problems, but not all pip users are familiar with Conda, or want to migrate to Conda just to use Read the Docs.

5.10.2 Security concerns

Builds are run in a Docker container, but the app controlling that container lives in the same server. Allowing to execute arbitrary commands with super user privileges may introduce some security issues.

5.10.3 Exposing `apt install`

For the previous reasons we won't allow to execute arbitrary commands with root (yet), but instead allow only to install extra packages using `apt`.

We would expose this through the config file. Users will provide a list of packages to install, and under the hook we would run:

- `apt update -y`
- `apt install -y {packages}`

These commands will be run before the Python setup step and after the clone step.

Note: All package names must be validated to avoid injection of extra options (like `-v`).

5.10.4 Using `docker exec`

Currently we use `docker exec` to execute commands in a running container. This command also allows to pass a user which is used to run the commands (#8058_). We can run the `apt` commands in our current containers using a super user momentarily.

5.10.5 Config file

The config file can add an additional mapping `build.apt_packages` to a list of packages to install.

```yaml
version: 2

build:
  apt_packages:
     - cmatrix
     - mysql-server
```

Note: Other names that were considered were:

- `build.packages`
- `build.extra_packages`
- `build.system_packages`

These were rejected to avoid confusion with existing keys, and to be explicit about the type of package.

5.10.6 Possible problems

- Some users may require to pass some additional flags or install from a ppa.
- Some packages may require some additional setup after installation.

5.10.7 Other possible solutions

- We can allow to run the containers as root doing something similar to what Travis does: They have one tool to convert the config file to a shell script (travis-build), and another that spins a docker container, executes that shell script and streams the logs back (travis-worker).
- A similar solution could be implemented using AWS Lambda.

This of course would require a large amount of work, but may be useful for the future.

5.11 Read the Docs data passed to Sphinx build context

Before calling `sphinx-build` to render your docs, Read the Docs injects some extra context in the templates by using the html_context Sphinx setting in the `conf.py` file. This extra context can be used to build some awesome features in your own theme.

Warning: This design document details future features that are **not yet implemented**. To discuss this document, please get in touch in the issue tracker.

Note: The Read the Docs Sphinx Theme uses this context to add additional features to the built documentation.

5.11.1 Context injected

Here is the full list of values injected by Read the Docs as a Python dictionary. Note that this dictionary is injected under the main key `readthedocs`:

```
{
    'readthedocs': {
        'v1': {
            'version': {
                'id': int,
                'slug': str,
                'verbose_name': str,
                'identifier': str,
                'type': str,
                'build_date': str,
                'downloads': {
                    'pdf': str,
                    'htmlzip': str,
                    'epub': str
                },
                'links': [{
                    'href': 'https://readthedocs.org/api/v2/version/{id}/',
                    'rel': 'self'
                }],
            },
            'project': {
                'id': int,
                'name': str,
                'slug': str,
                'description': str,
                'language': str,
                'canonical_url': str,
                'subprojects': [{
                    'id': int,
                    'name': str,
                    'slug': str,
                    'description': str,
```

(continues on next page)

(continued from previous page)

```
                    'language': str,
                    'canonical_url': str,
                    'links': [{
                        'href': 'https://readthedocs.org/api/v2/project/{id}/',
                        'rel': 'self'
                    }]
                }],
                'links': [{
                    'href': 'https://readthedocs.org/api/v2/project/{id}/',
                    'rel': 'self'
                }]
            },
            'sphinx': {
                'html_theme': str,
                'source_suffix': str
            },
            'analytics': {
                'user_analytics_code': str,
                'global_analytics_code': str
            },
            'vcs': {
                'type': str,  # 'bitbucket', 'github', 'gitlab' or 'svn'
                'user': str,
                'repo': str,
                'commit': str,
                'version': str,
                'display': bool,
                'conf_py_path': str
            },
            'meta': {
                'API_HOST': str,
                'MEDIA_URL': str,
                'PRODUCTION_DOMAIN': str,
                'READTHEDOCS': True
            }
        }
    }
}
```

Warning: Read the Docs passes information to `sphinx-build` that may change in the future (e.g. at the moment of building the version `0.6` this was the `latest` but then `0.7` and `0.8` were added to the project and also built under Read the Docs) so it's your responsibility to use this context in a proper way.

In case you want *fresh data* at the moment of reading your documentation, you should consider using the Read the Docs Public API via Javascript.

5.11.2 Using Read the Docs context in your theme

In case you want to access to this data from your theme, you can use it like this:

```
{% if readthedocs.v1.vcs.type == 'github' %}
    <a href="https://github.com/{{ readthedocs.v1.vcs.user }}/{{ readthedocs.v1.vcs.repo␣
↪}}
    /blob/{{ readthedocs.v1.vcs.version }}{{ readthedocs.v1.vcs.conf_py_path }}{{␣
↪pagename }}.rst">
    Show on GitHub</a>
{% endif %}
```

Note: In this example, we are using `pagename` which is a Sphinx variable representing the name of the page you are on. More information about Sphinx variables can be found in the Sphinx documentation.

5.11.3 Customizing the context

In case you want to add some extra context you will have to declare your own `html_context` in your `conf.py` like this:

```
html_context = {
    'author': 'My Name',
    'date': datetime.date.today().strftime('%d/%m/%y'),
}
```

and use it inside your theme as:

```
<p>This documentation was written by {{ author }} on {{ date }}.</p>
```

Warning: Take into account that the Read the Docs context is injected after your definition of `html_context` so, it's not possible to override Read the Docs context values.

5.12 YAML Configuration File

5.12.1 Background

The current YAML configuration file is in beta state. There are many options and features that it doesn't support yet. This document will serve as a design document for discuss how to implement the missing features.

5.12.2 Scope

- Finish the spec to include all the missing options
- Have consistency around the spec
- Proper documentation for the end user
- Allow to specify the spec's version used on the YAML file
- Collect/show metadata about the YAML file and build configuration
- Promote the adoption of the configuration file

5.12.3 RTD settings

No all the RTD settings are applicable to the YAML file, others are applicable for each build (or version), and others for the global project.

Not applicable settings

Those settings can't be on the YAML file because: may depend for the initial project setup, are planned to be removed, security and privacy reasons.

- Project Name
- Repo URL
- Repo type
- Privacy level (this feature is planned to be removed[1])
- Project description (this feature is planned to be removed[2])
- Single version
- Default branch
- Default version
- Domains
- Active versions
- Translations
- Subprojects
- Integrations
- Notifications
- Language
- Programming Language
- Project homepage
- Tags
- Analytics code
- Global redirects

[1] https://github.com/readthedocs/readthedocs.org/issues/2663
[2] https://github.com/readthedocs/readthedocs.org/issues/3689

Global settings

To keep consistency with the per-version settings and avoid confusion, this settings will not be stored in the YAML file and will be stored in the database only.

Local settings

Those configurations will be read from the YAML file in the current version that is being built.

Several settings are already implemented and documented on https://docs.readthedocs.io/en/latest/yaml-config.html. So, they aren't covered with much detail here.

- Documentation type
- Project installation (virtual env, requirements file, sphinx configuration file, etc)
- Additional builds (pdf, epub)
- Python interpreter
- Per-version redirects

5.12.4 Configuration file

Format

The file format is based on the YAML spec 1.2[3] (latest version on the time of this writing).

The file must be on the root directory of the repository, and must be named as:

- `readthedocs.yml`
- `readthedocs.yaml`
- `.readthedocs.yml`
- `.readthedocs.yaml`

Conventions

The spec of the configuration file must use this conventions.

- Use `[]` to indicate an empty list
- Use `null` to indicate a null value
- Use `all` (internal string keyword) to indicate that all options are included on a list with predetermined choices.
- Use `true` and `false` as only options on boolean fields

[3] https://yaml.org/spec/1.2/spec.html

Spec

The current spec is documented on https://docs.readthedocs.io/en/latest/yaml-config.html. It will be used as base for the future spec. The spec will be written using a validation schema such as https://json-schema-everywhere.github.io/yaml.

5.12.5 Versioning the spec

The version of the spec that the user wants to use will be specified on the YAML file. The spec only will have mayor versions (1.0, not 1.2)[4]. For keeping compatibility with older projects using a configuration file without a version, the latest compatible version will be used (1.0).

5.12.6 Adoption of the configuration file

When a user creates a new project or it's on the settings page, we could suggest her/him an example of a functional configuration file with a minimal setup. And making clear where to put global configurations.

For users that already have a project, we can suggest him/her a configuration file on each build based on the current settings.

5.12.7 Configuration file and database

The settings used in the build from the configuration file (and other metadata) needs to be stored in the database, this is for later usage only, not to populate existing fields.

5.12.8 The build process

- The repository is updated
- Checkout to the current version
- Retrieve the settings from the database
- Try to parse the YAML file (the build fails if there is an error)
- Merge the both settings (YAML file and database)
- The version is built according to the settings
- The settings used to build the documentation can be seen by the user

5.12.9 Dependencies

Current repository which contains the code related to the configuration file: https://github.com/readthedocs/readthedocs-build

[4] https://github.com/readthedocs/readthedocs.org/issues/3806

5.12.10 Footnotes

CHAPTER

SIX

DEVELOPMENT INSTALLATION

These are development setup and *standards* that are adhered to by the core development team while developing Read the Docs and related services. If you are a contributor to Read the Docs, it might a be a good idea to follow these guidelines as well.

To follow these instructions you will need a Unix-like operating system, or Windows Subsystem for Linux (WSL). Other operating systems are not supported.

Note: We do not recommend to follow this guide to deploy an instance of Read the Docs for production usage. Take into account that this setup is only useful for developing purposes.

6.1 Set up your environment

1. install Docker following their installation guide.
2. clone the `readthedocs.org` repository:

   ```
   $ git clone --recurse-submodules https://github.com/readthedocs/readthedocs.org/
   ```

3. install the requirements from `common` submodule:

   ```
   $ pip install -r common/dockerfiles/requirements.txt
   ```

4. build the Docker image for the servers:

   Warning: This command could take a while to finish since it will download several Docker images.

   ```
   $ inv docker.build
   ```

   Tip: If you pass `GITHUB_TOKEN` environment variable to this command, it will add support for readthedocs-ext.

5. pull down Docker images for the builders:

   ```
   $ inv docker.pull --only-latest
   ```

6. start all the containers:

   ```
   $ inv docker.up  --init  # --init is only needed the first time
   ```

7. add read permissions to the storage backend:

   - go to http://localhost:9000/ (MinIO S3 storage backend)

- login as `admin` / `password`
- click "…" next to the `static` bucket name and then "Edit Policy"
- leave "prefix" empty and click "Add" to give "Read Only" access on the `static` bucket
- click on the "+" icon on the bottom-right corner, then "Create bucket" with the name `media`, hit Enter on the keyboard, and repeat the operation above to give "Read Only" access to it

8. go to http://community.dev.readthedocs.io to access your local instance of Read the Docs.

6.2 Check that everything works

1. go to http://community.dev.readthedocs.io and check that the appearance and style looks correct (otherwise the MinIO buckets might be misconfigured, see above)
2. login as `admin` / `admin` and verify that the project list appears
3. go to the "Read the Docs" project, click on the "Build version" button to build `latest`, and wait until it finishes
4. click on the "View docs" button to browse the documentation, and verify that it works

6.3 Working with Docker Compose

We wrote a wrapper with `invoke` around `docker-compose` to have some shortcuts and save some work while typing docker compose commands. This section explains these `invoke` commands:

`inv docker.build` Builds the generic Docker image used by our servers (web, celery, build and proxito).

`inv docker.up` Starts all the containers needed to run Read the Docs completely.

- `--no-search` can be passed to disable search
- `--init` is used the first time this command is ran to run initial migrations, create an admin user, etc
- `--no-reload` makes all celery processes and django runserver to use no reload and do not watch for files changes

`inv docker.shell` Opens a shell in a container (web by default).

- `--no-running` spins up a new container and open a shell
- `--container` specifies in which container the shell is open

`inv docker.manage {command}` Executes a Django management command in a container.

Tip: Useful when modifying models to run `makemigrations`.

`inv docker.down` Stops and removes all containers running.

- `--volumes` will remove the volumes as well (database data will be lost)

`inv docker.restart {containers}` Restarts the containers specified (automatically restarts NGINX when needed).

`inv docker.attach {container}` Grab STDIN/STDOUT control of a running container.

Tip: Useful to debug with `pdb`. Once the program has stopped in your pdb line, you can run `inv docker.attach web` and jump into a pdb session (it also works with ipdb and pdb++)

Tip: You can hit CTRL-p CTRL-p to detach it without stopping the running process.

`inv docker.test` Runs all the test suites inside the container.

- `--arguments` will pass arguments to Tox command (e.g. `--arguments "-e py38 -- -k test_api"`)

`inv docker.pull` Downloads and tags all the Docker images required for builders.

- `--only-latest` does not pull `stable` and `testing` images.

`inv docker.buildassets` Build all the assets and "deploy" them to the storage.

6.3.1 Adding a new Python dependency

The Docker image for the servers is built with the requirements defined in the current checked out branch. In case you need to add a new Python dependency while developing, you can use the `common/dockerfiles/entrypoints/common.sh` script as shortcut.

This script is run at startup on all the servers (web, celery, builder, proxito) which allows you to test your dependency without re-building the whole image. To do this, add the `pip` command required for your dependency in `common.sh` file:

```
# common.sh
pip install my-dependency==1.2.3
```

Once the PR that adds this dependency was merged, you can rebuild the image so the dependency is added to the Docker image itself and it's not needed to be installed each time the container spins up.

6.3.2 Debugging Celery

In order to step into the worker process, you can't use `pdb` or `ipdb`, but you can use `celery.contrib.rdb`:

```
from celery.contrib import rdb; rdb.set_trace()
```

When the breakpoint is hit, the Celery worker will pause on the breakpoint and will alert you on STDOUT of a port to connect to. You can open a shell into the container with `inv docker.shell celery` (or `build`) and then use `telnet` or `netcat` to connect to the debug process port:

```
$ nc 127.0.0.1 6900
```

The `rdb` debugger is similar to `pdb`, there is no `ipdb` for remote debugging currently.

6.3.3 Configuring connected accounts

These are optional steps to setup the connected accounts (GitHub, GitLab, and BitBucket) in your development environment. This will allow you to login to your local development instance using your GitHub, Bitbucket, or GitLab credentials and this makes the process of importing repositories easier.

However, because these services will not be able to connect back to your local development instance, incoming webhooks will not function correctly. For some services, the webhooks will fail to be added when the repository is imported. For others, the webhook will simply fail to connect when there are new commits to the repository.

- Configure the applications on GitHub, Bitbucket, and GitLab. For each of these, the callback URI is `http://community.dev.readthedocs.io/accounts/<provider>/login/callback/` where `<provider>` is one of `github`, `gitlab`, or `bitbucket_oauth2`. When setup, you will be given a "Client ID" (also called an "Application ID" or just "Key") and a "Secret".
- Take the "Client ID" and "Secret" for each service and enter it in your local Django admin at: `http://community.dev.readthedocs.io/admin/socialaccount/socialapp/`. Make sure to apply it to the "Site".

6.4 Core team standards

Core team members expect to have a development environment that closely approximates our production environment, in order to spot bugs and logical inconsistencies before they make their way to production.

This solution gives us many features that allows us to have an environment closer to production:

Celery runs as a separate process Avoids masking bugs that could be introduced by Celery tasks in a race conditions.

Celery runs multiple processes We run celery with multiple worker processes to discover race conditions between tasks.

Docker for builds Docker is used for a build backend instead of the local host build backend. There are a number of differences between the two execution methods in how processes are executed, what is installed, and what can potentially leak through and mask bugs – for example, local SSH agent allowing code check not normally possible.

Serve documentation under a subdomain There are a number of resolution bugs and cross-domain behavior that can only be caught by using `USE_SUBDOMAIN` setting.

PostgreSQL as a database It is recommended that Postgres be used as the default database whenever possible, as SQLite has issues with our Django version and we use Postgres in production. Differences between Postgres and SQLite should be masked for the most part however, as Django does abstract database procedures, and we don't do any Postgres-specific operations yet.

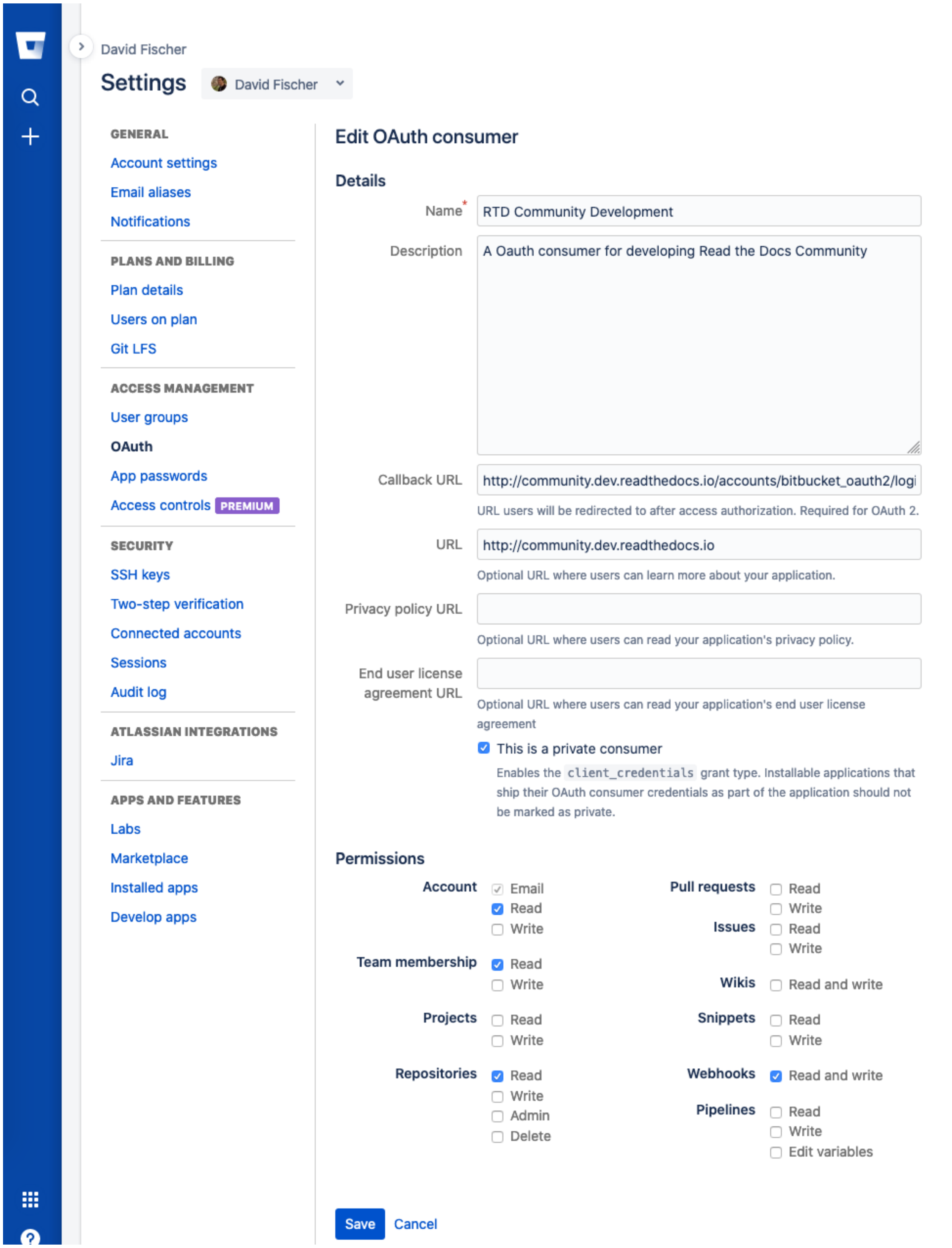
Celery is isolated from database Celery workers on our build servers do not have database access and need to be written to use API access instead.

Use NGINX as web server All the site is served via NGINX with the ability to change some configuration locally.

MinIO as Django storage backend All static and media files are served using Minio –an emulator of S3, which is the one used in production.

Serve documentation via El Proxito Documentation is proxied by NGINX to El Proxito and proxied back to NGINX to be served finally. El Proxito is a small application put in front of the documentation to serve files from the Django Storage Backend.

Search enabled by default Elasticsearch is properly configured and enabled by default. All the documentation indexes are updated after a build is finished.



Edit OAuth consumer

Details

Name*: RTD Community Development

Description: A Oauth consumer for developing Read the Docs Community

Callback URL: http://community.dev.readthedocs.io/accounts/bitbucket_oauth2/logi

URL users will be redirected to after access authorization. Required for OAuth 2.

URL: http://community.dev.readthedocs.io

Optional URL where users can learn more about your application.

Privacy policy URL:

Optional URL where users can read your application's privacy policy.

End user license agreement URL:

Optional URL where users can read your application's end user license agreement

[✓] This is a private consumer

Enables the `client_credentials` grant type. Installable applications that ship their OAuth consumer credentials as part of the application should not be marked as private.

Permissions

Account	[✓] Email [✓] Read [ ] Write	Pull requests	[ ] Read [ ] Write
		Issues	[ ] Read [ ] Write
Team membership	[✓] Read [ ] Write	Wikis	[ ] Read and write
Projects	[ ] Read [ ] Write	Snippets	[ ] Read [ ] Write
Repositories	[✓] Read [ ] Write [ ] Admin [ ] Delete	Webhooks	[✓] Read and write
		Pipelines	[ ] Read [ ] Write [ ] Edit variables

Fig. 1: Configuring an OAuth consumer for local development on Bitbucket

CHAPTER

SEVEN

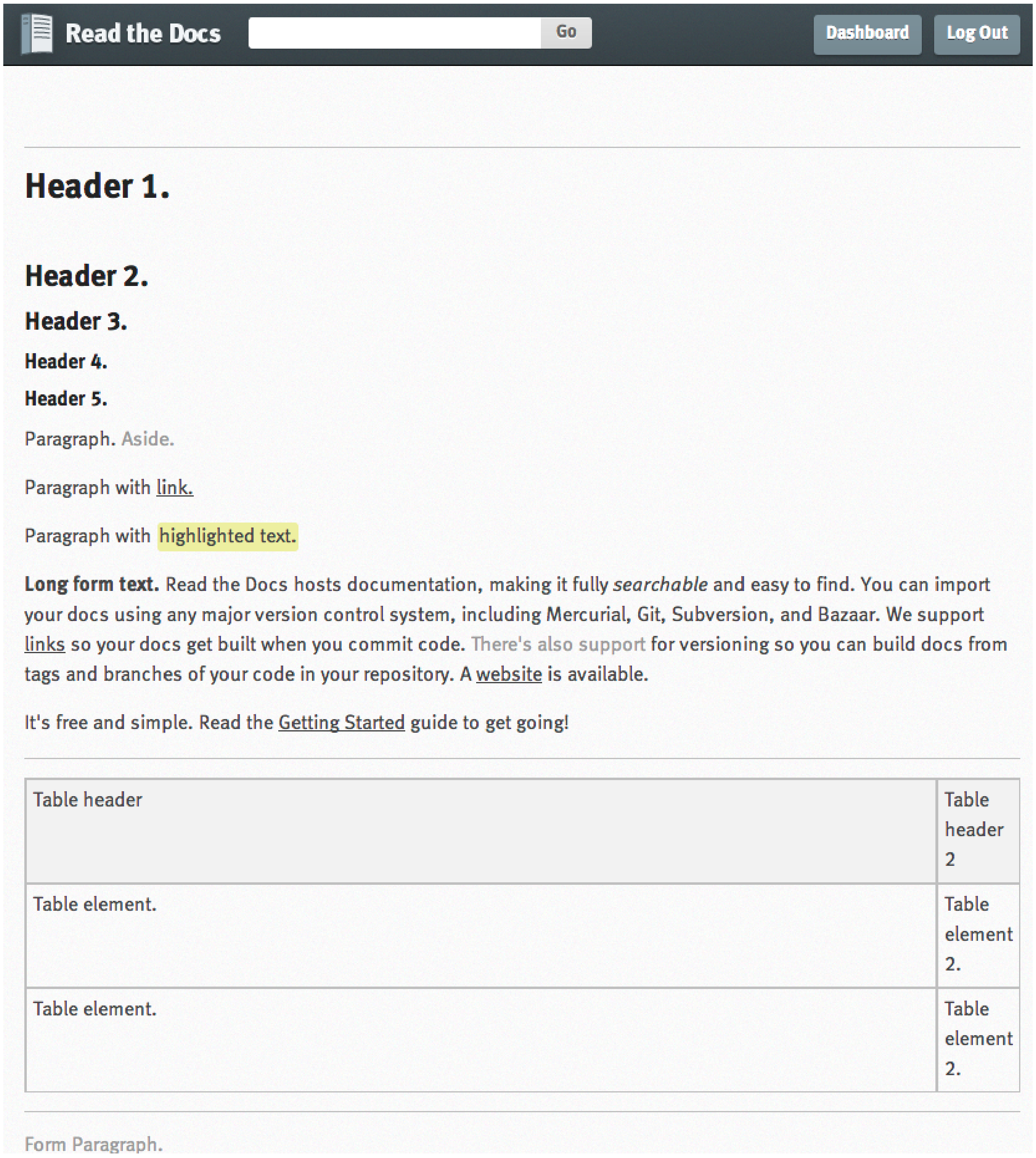
DESIGNING READ THE DOCS

So you're thinking of contributing some of your time and design skills to Read the Docs? That's **awesome**. This document will lead you through a few features available to ease the process of working with Read the Doc's CSS and static assets.

To start, you should follow the *Development Installation* instructions to get a working copy of the Read the Docs repository locally.

7.1 Style Catalog

Once you have RTD running locally, you can open `http://localhost:8000/style-catalog/` for a quick overview of the currently available styles.



Header 1.

Header 2.

Header 3.

Header 4.

Header 5.

Paragraph. Aside.

Paragraph with link.

Paragraph with highlighted text.

Long form text. Read the Docs hosts documentation, making it fully *searchable* and easy to find. You can import your docs using any major version control system, including Mercurial, Git, Subversion, and Bazaar. We support links so your docs get built when you commit code. There's also support for versioning so you can build docs from tags and branches of your code in your repository. A website is available.

It's free and simple. Read the Getting Started guide to get going!

Table header	Table header 2
Table element.	Table element 2.
Table element.	Table element 2.

Form Paragraph.

This way you can quickly get started writing HTML – or if you're modifying existing styles you can get a quick idea of how things will change site-wide.

7.2 Readthedocs.org Changes

Styles for the primary RTD site are located in `media/css` directory.

These styles only affect the primary site – **not** any of the generated documentation using the default RTD style.

7.3 Contributing

Contributions should follow the *Contributing to Read the Docs* guidelines where applicable – ideally you'll create a pull request against the Read the Docs GitHub project from your forked repo and include a brief description of what you added / removed / changed, as well as an attached image (you can just take a screenshot and drop it into the PR creation form) of the effects of your changes.

There's not a hard browser range, but your design changes should work reasonably well across all major browsers, IE8+ – that's not to say it needs to be pixel-perfect in older browsers! Just avoid making changes that render older browsers utterly unusable (or provide a sane fallback).

7.4 Brand Guidelines

Find our branding guidelines in our guidelines documentation: https://read-the-docs-guidelines.readthedocs-hosted.com.

CHAPTER

EIGHT

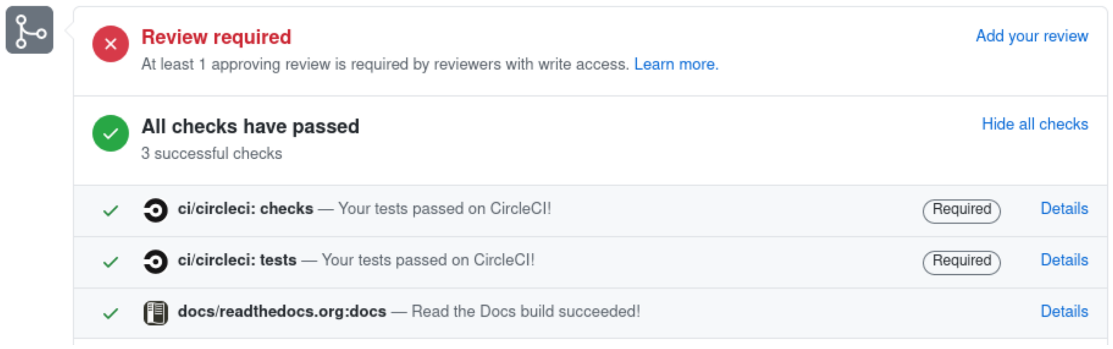
BUILDING AND CONTRIBUTING TO DOCUMENTATION

As one might expect, the documentation for Read the Docs is built using Sphinx and hosted on Read the Docs. The docs are kept in the `docs/` directory at the top of the source tree, and are divided into developer and user-facing documentation.

8.1 Contributing through the Github UI

If you're making small changes to the documentation, you can verify those changes through the documentation generated when you open a PR and can be accessed using the Github UI.

1. click the checkmark next to your commit and it will expand to have multiple options
2. click the "details" link next to the "docs/readthedocs.org:docs" item



Review required

At least 1 approving review is required by reviewers with write access. Learn more.

Add your review

All checks have passed

3 successful checks

Hide all checks

ci/circleci: checks — Your tests passed on CircleCI! Required Details

ci/circleci: tests — Your tests passed on CircleCI! Required Details

docs/readthedocs.org:docs — Read the Docs build succeeded! Details

3. navigate to the section of the documentation you worked on to verify your changes

8.2 Contributing from your local machine

If you're making large changes to the documentation, you may want to verify those changes locally before pushing upstream.

1. clone the `readthedocs.org` repository:

```
$ git clone --recurse-submodules https://github.com/readthedocs/readthedocs.org/
```

2. create a virtual environment with Python 3.8 (preferably the latest release, 3.8.12 at the time of writing), activate it, and upgrade pip:

```
$ cd readthedocs.org
$ python3.8 -m venv .venv
$ source .venv/bin/activate
(.venv) $ python -m pip install -U pip
```

3. install documentation requirements

```
(.venv) $ pip install -r requirements/testing.txt
(.venv) $ pip install -r requirements/docs.txt
```

4. build the documents

   To build the user-facing documentation:

```
(.venv) $ cd docs
(.venv) $ make livehtml
```

   To build the developer documentation:

```
(.venv) $ cd docs
(.venv) $ RTD_DOCSET=dev make livehtml
```

5. the documents will be available at http://127.0.0.1:4444/ and will rebuild each time you edit and save a file.

8.3 Guidelines

Please follow these guidelines when updating our docs. Let us know if you have any questions or something isn't clear.

8.3.1 The brand

We are called **Read the Docs**. The *the* is not capitalized.

We do however use the acronym **RTD**.

8.3.2 Titles

For page titles, or Heading1 as they are sometimes called, we use title-case.

If the page includes multiple sub-headings (H2, H3), we usually use sentence-case unless the titles include terminology that is supposed to be capitalized.

8.3.3 Content

- Do not break the content across multiple lines at 80 characters, but rather break them on semantic meaning (e.g. periods or commas). Read more about this here.
- If you are cross-referencing to a different page within our website, use the `doc` role and not a hyperlink.
- If you are cross-referencing to a section within our website, use the `ref` role with the label from the autosection-label extension.

CHAPTER

NINE

FRONT END DEVELOPMENT

9.1 Background

Note: Consider this the canonical resource for contributing Javascript and CSS. We are currently in the process of modernizing our front end development procedures. You will see a lot of different styles around the code base for front end JavaScript and CSS.

Our modern front end development stack includes the following tools:

- Gulp
- Bower
- Browserify
- Debowerify
- And soon, LESS

We use the following libraries:

- Knockout
- jQuery
- Several jQuery plugins

Previously, JavaScript development has been done in monolithic files or inside templates. jQuery was added as a global object via an include in the base template to an external source. There are no standards currently to JavaScript libraries, this aims to solve that.

The requirements for modernizing our front end code are:

- Code should be modular and testable. One-off chunks of JavaScript in templates or in large monolithic files are not easily testable. We currently have no JavaScript tests.
- Reduce code duplication.
- Easy JavaScript dependency management.

Modularizing code with Browserify is a good first step. In this development workflow, major dependencies commonly used across JavaScript includes are installed with Bower for testing, and vendorized as standalone libraries via Gulp and Browserify. This way, we can easily test our JavaScript libraries against jQuery/etc, and have the flexibility of modularizing our code. See *JavaScript Bundles* for more information on what and how we are bundling.

To ease deployment and contributions, bundled JavaScript is checked into the repository for now. This ensures new contributors don't need an additional front end stack just for making changes to our Python code base. In the future, this may change, so that assets are compiled before deployment, however as our front end assets are in a state of flux, it's easier to keep absolute sources checked in.

9.2 Getting Started

You will need to follow our *guide to install a development Read the Docs instance* first.

The sources for our bundles are found in the per-application path `static-src`, which has the same directory structure as `static`. Files in `static-src` are compiled to `static` for static file collection in Django. Don't edit files in `static` directly, unless you are sure there isn't a source file that will compile over your changes.

To compile your changes and make them available in the application you need to run:

```
inv docker.buildassets
```

Once you are happy with your changes, make sure to check in both files under `static` and `static-src`, and commit those.

9.3 Making Changes

If you are creating a new library, or a new library entry point, make sure to define the application source file in `gulpfile.js`, this is not handled automatically right now.

If you are bringing in a new vendor library, make sure to define the bundles you are going to create in `gulpfile.js` as well.

Tests should be included per-application, in a path called `tests`, under the `static-src/js` path you are working in. Currently, we still need a test runner that accumulates these files.

9.4 Deployment

If merging several branches with JavaScript changes, it's important to do a final post-merge bundle. Follow the steps above to rebundle the libraries, and check in any changed libraries.

9.5 JavaScript Bundles

There are several components to our bundling scheme:

Vendor library We repackage these using Browserify, Bower, and Debowerify to make these libraries available by a `require` statement. Vendor libraries are packaged separately from our JavaScript libraries, because we use the vendor libraries in multiple locations. Libraries bundled this way with Browserify are available to our libraries via `require` and will back down to finding the object on the global `window` scope.

Vendor libraries should only include libraries we are commonly reusing. This currently includes `jQuery` and `Knockout`. These modules will be excluded from libraries by special includes in our `gulpfile.js`.

Minor third party libraries These libraries are maybe used in one or two locations. They are installed via Bower and included in the output library file. Because we aren't reusing them commonly, they don’t require a separate bundle or separate include. Examples here would include jQuery plugins used on one off forms, such as jQuery Payments.

Our libraries These libraries are bundled up excluding vendor libraries ignored by rules in our `gulpfile.js`. These files should be organized by function and can be split up into multiple files per application.

Entry points to libraries must be defined in `gulpfile.js` for now. We don’t have a defined directory structure that would make it easy to imply the entry point to an application library.

CHAPTER

TEN

INTERNATIONALIZATION

This document covers the details regarding internationalization and localization that are applied in Read the Docs. The guidelines described are mostly based on Kitsune's localization documentation.

As with most of the Django applications out there, Read the Docs' i18n/l10n framework is based on GNU gettext. Crowd-sourced localization is optionally available at Transifex.

For more information about the general ideas, look at this document: http://www.gnu.org/software/gettext/manual/html_node/Concepts.html

10.1 Making Strings Localizable

Making strings in templates localizable is exceptionally easy. Making strings in Python localizable is a little more complicated. The short answer, though, is to just wrap the string in `_()`.

10.1.1 Interpolation

A string is often a combination of a fixed string and something changing, for example, `Welcome, James` is a combination of the fixed part `Welcome,`, and the changing part `James`. The naive solution is to localize the first part and then follow it with the name:

```
_('Welcome, ') + username
```

This is **wrong!**

In some locales, the word order may be different. Use Python string formatting to interpolate the changing part into the string:

```
_('Welcome, {name}').format(name=username)
```

Python gives you a lot of ways to interpolate strings. The best way is to use Py3k formatting and kwargs. That's the clearest for localizers.

10.1.2 Localization Comments

Sometimes, it can help localizers to describe where a string comes from, particularly if it can be difficult to find in the interface, or is not very self-descriptive (e.g. very short strings). If you immediately precede the string with a comment that starts with `Translators:`, the comment will be added to the PO file, and visible to localizers.

Example:

```
DEFAULT_THEME_CHOICES = (
    # Translators: This is a name of a Sphinx theme.
    (THEME_DEFAULT, _('Default')),
    # Translators: This is a name of a Sphinx theme.
    (THEME_SPHINX, _('Sphinx Docs')),
    # Translators: This is a name of a Sphinx theme.
    (THEME_TRADITIONAL, _('Traditional')),
    # Translators: This is a name of a Sphinx theme.
    (THEME_NATURE, _('Nature')),
    # Translators: This is a name of a Sphinx theme.
    (THEME_HAIKU, _('Haiku')),
)
```

10.1.3 Adding Context with msgctxt

Strings may be the same in English, but different in other languages. English, for example, has no grammatical gender, and sometimes the noun and verb forms of a word are identical.

To make it possible to localize these correctly, we can add “context” (known in gettext as *msgctxt*) to differentiate two otherwise identical strings. Django provides a `pgettext()` function for this.

For example, the string *Search* may be a noun or a verb in English. In a heading, it may be considered a noun, but on a button, it may be a verb. It’s appropriate to add a context (like *button*) to one of them.

Generally, we should only add context if we are sure the strings aren’t used in the same way, or if localizers ask us to.

Example:

```
from django.utils.translation import pgettext

month = pgettext("text for the search button on the form", "Search")
```

10.1.4 Plurals

You have 1 new messages grates on discerning ears. Fortunately, gettext gives us a way to fix that in English *and* other locales, the `ngettext()` function:

```
ngettext('singular sentence', 'plural sentence', count)
```

A more realistic example might be:

```
ngettext('Found {count} result.',
         'Found {count} results',
         len(results)).format(count=len(results))
```

This method takes three arguments because English only needs three, i.e., zero is considered "plural" for English. Other languages may have different plural rules, and require different phrases for, say 0, 1, 2-3, 4-10, >10. That's absolutely fine, and gettext makes it possible.

10.2 Strings in Templates

When putting new text into a template, all you need to do is wrap it in a `{% trans %}` template tag:

```
<h1>{% trans "Heading" %}</h1>
```

Context can be added, too:

```
<h1>{% trans "Heading" context "section name" %}</h1>
```

Comments for translators need to precede the internationalized text and must start with the `Translators:` keyword.:

```
{# Translators: This heading is displayed in the user's profile page #}
<h1>{% trans "Heading" %}</h1>
```

To interpolate, you need to use the alternative and more verbose `{% blocktrans %}` template tag — it's actually a block:

```
{% blocktrans %}Welcome, {{ name }}!{% endblocktrans %}
```

Note that the `{{ name }}` variable needs to exist in the template context.

In some situations, it's desirable to evaluate template expressions such as filters or accessing object attributes. You can't do that within the `{% blocktrans %}` block, so you need to bind the expression to a local variable first:

```
{% blocktrans trimmed with revision.created_date|timesince as timesince %}
{{ revision }} {{ timesince }} ago
{% endblocktrans %}

{% blocktrans with project.name as name %}Delete {{ name }}?{% endblocktrans %}
```

`{% blocktrans %}` also provides pluralization. For that you need to bind a counter with the name `count` and provide a plural translation after the `{% plural %}` tag:

```
{% blocktrans trimmed with amount=article.price count years=i.length %}
That will cost $ {{ amount }} per year.
{% plural %}
That will cost $ {{ amount }} per {{ years }} years.
{% endblocktrans %}
```

Note: The previous multi-lines examples also use the `trimmed` option. This removes newline characters and replaces any whitespace at the beginning and end of a line, helping translators when translating these strings.

10.3 Strings in Python

Note: Whenever you are adding a string in Python, ask yourself if it really needs to be there, or if it should be in the template. Keep logic and presentation separate!

Strings in Python are more complex for two reasons:

1. We need to make sure we're always using Unicode strings and the Unicode-friendly versions of the functions.
2. If you use the `gettext()` function in the wrong place, the string may end up in the wrong locale!

Here's how you might localize a string in a view:

```
from django.utils.translation import gettext as _

def my_view(request):
    if request.user.is_superuser:
        msg = _(u'Oh hi, staff!')
    else:
        msg = _(u'You are not staff!')
```

Interpolation is done through normal Python string formatting:

```
msg = _(u'Oh, hi, {user}').format(user=request.user.username)
```

Context information can be supplied by using the `pgettext()` function:

```
msg = pgettext('the context', 'Search')
```

Translator comments are normal one-line Python comments:

```
# Translators: A message to users.
msg = _(u'Oh, hi there!')
```

If you need to use plurals, import the `ungettext()` function:

```
from django.utils.translation import ungettext

n = len(results)
msg = ungettext('Found {0} result', 'Found {0} results', n).format(n)
```

10.3.1 Lazily Translated Strings

You can use `gettext()` or `ungettext()` only in views or functions called from views. If the function will be evaluated when the module is loaded, then the string may end up in English or the locale of the last request!

Examples include strings in module-level code, arguments to functions in class definitions, strings in functions called from outside the context of a view. To internationalize these strings, you need to use the `_lazy` versions of the above methods, `gettext_lazy()` and `ungettext_lazy()`. The result doesn't get translated until it is evaluated as a string, for example by being output or passed to `unicode()`:

```
from django.utils.translation import gettext_lazy as _

class UserProfileForm(forms.ModelForm):
    first_name = CharField(label=_('First name'), required=False)
    last_name = CharField(label=_('Last name'), required=False)
```

In case you want to provide context to a lazily-evaluated gettext string, you will need to use `pgettext_lazy()`.

10.4 Administrative Tasks

10.4.1 Updating Localization Files

To update the translation source files (eg if you changed or added translatable strings in the templates or Python code) you should run `python manage.py makemessages -l <language>` in the project's root directory (substitute `<language>` with a valid language code).

The updated files can now be localized in a PO editor or crowd-sourced online translation tool.

10.4.2 Compiling to MO

Gettext doesn't parse any text files, it reads a binary format for faster performance. To compile the latest PO files in the repository, Django provides the `compilemessages` management command. For example, to compile all the available localizations, just run:

```
$ python manage.py compilemessages -a
```

You will need to do this every time you want to push updated translations to the live site.

Also, note that it's not a good idea to track MO files in version control, since they would need to be updated at the same pace PO files are updated, so it's silly and not worth it. They are ignored by `.gitignore`, but please make sure you don't forcibly add them to the repository.

10.4.3 Transifex Integration

To push updated translation source files to Transifex, run `tx push -s` (for English) or `tx push -t <language>` (for non-English).

To pull changes from Transifex, run `tx pull -a`. Note that Transifex does not compile the translation files, so you have to do this after the pull (see the *Compiling to MO* section).

For more information about the `tx` command, read the Transifex client's help pages.

Note: For the Read the Docs community site, we use Invoke with a tasks.py file to follow this process:

1. Update files and push sources (English) to Transifex:

   ```
   $ invoke l10n.push
   ```

2. Pull the updated translations from Transifex:

   ```
   $ invoke l10n.pull
   ```

CHAPTER

ELEVEN

SEARCH

Read The Docs uses Elasticsearch instead of the built in Sphinx search for providing better search results. Documents are indexed in the Elasticsearch index and the search is made through the API. All the Search Code is open source and lives in the GitHub Repository. Currently we are using Elasticsearch 6.3.

11.1 Local Development Configuration

Elasticsearch is installed and run as part of the *development installation guide*.

11.1.1 Indexing into Elasticsearch

For using search, you need to index data to the Elasticsearch Index. Run `reindex_elasticsearch` management command:

```
$ inv docker.manage reindex_elasticsearch
```

For performance optimization, we implemented our own version of management command rather than the built in management command provided by the django-elasticsearch-dsl package.

11.1.2 Auto Indexing

By default, Auto Indexing is turned off in development mode. To turn it on, change the `ELASTICSEARCH_DSL_AUTOSYNC` settings to `True` in the `readthedocs/settings/dev.py` file. After that, whenever a documentation successfully builds, or project gets added, the search index will update automatically.

11.2 Architecture

The search architecture is divided into 2 parts.

- One part is responsible for **indexing** the documents and projects (`documents.py`)
- The other part is responsible for **querying** the Index to show the proper results to users (`faceted_search.py`)

We use the django-elasticsearch-dsl package for our Document abstraction. django-elasticsearch-dsl is a wrapper around elasticsearch-dsl for easy configuration with Django.

11.2.1 Indexing

All the Sphinx documents are indexed into Elasticsearch after the build is successful. Currently, we do not index MkDocs documents to elasticsearch, but any kind of help is welcome.

11.2.2 Troubleshooting

If you get an error like:

```
RequestError(400, 'search_phase_execution_exception', 'failed to create query: ...
```

You can fix this by deleting the page index:

```
$ inv docker.manage 'search_index --delete'
```

Note: You'll need to *reindex the projects* after this.

How we index documentations

After any build is successfully finished, `HTMLFile` objects are created for each of the `HTML` files and the old version's `HTMLFile` object is deleted. By default, django-elasticsearch-dsl package listens to the `post_create`/`post_delete` signals to index/delete documents, but it has performance drawbacks as it send HTTP request whenever any `HTMLFile` objects is created or deleted. To optimize the performance, `bulk_post_create` and `bulk_post_delete` signals are dispatched with list of `HTMLFIle` objects so its possible to bulk index documents in elasticsearch (`bulk_post_create` signal is dispatched for created and `bulk_post_delete` is dispatched for deleted objects). Both of the signals are dispatched with the list of the instances of `HTMLFile` in `instance_list` parameter.

We listen to the `bulk_post_create` and `bulk_post_delete` signals in our `Search` application and index/delete the documentation content from the `HTMLFile` instances.

How we index projects

We also index project information in our search index so that the user can search for projects from the main site. We listen to the `post_create` and `post_delete` signals of `Project` model and index/delete into Elasticsearch accordingly.

Elasticsearch Document

elasticsearch-dsl provides a model-like wrapper for the Elasticsearch document. As per requirements of django-elasticsearch-dsl, it is stored in the `readthedocs/search/documents.py` file.

> **ProjectDocument:** It is used for indexing projects. Signal listener of django-elasticsearch-dsl listens to the `post_save` signal of `Project` model and then index/delete into Elasticsearch.
>
> **PageDocument**: It is used for indexing documentation of projects. As mentioned above, our `Search` app listens to the `bulk_post_create` and `bulk_post_delete` signals and indexes/deleted documentation into Elasticsearch. The signal listeners are in the `readthedocs/search/signals.py` file. Both of the signals are dispatched after a successful documentation build.
>
> The fields and ES Datatypes are specified in the `PageDocument`. The indexable data is taken from `processed_json` property of `HTMLFile`. This property provides python dictionary with document data like `title`, `sections`, `path` etc.

CHAPTER

TWELVE

SERVER SIDE SEARCH INTEGRATION

Read the Docs provides server side search (SSS) in replace of the default search engine of your site. To accomplish this, Read the Docs parses the content directly from your HTML pages*[0].

If you are the author of a theme or a static site generator you can read this document, and follow some conventions in order to improve the integration of SSS with your theme/site.

12.1 Indexing

The content of the page is parsed into sections, in general, the indexing process happens in three steps:

1. Identify the main content node.
2. Remove any irrelevant content from the main node.
3. Parse all sections inside the main node.

Read the Docs makes use of ARIA roles and other heuristics in order to process the content.

Tip: Following the ARIA conventions will also improve the accessibility of your site. See also https://webaim.org/techniques/semanticstructure/.

12.1.1 Main content node

The main content node should have a main role (or a `main` tag), and there should only be one per page. This node is the one that contains all the page content. Example:

```html
<html>
   <head>
      ...
   </head>
   <body>
      <div>
         This content isn't processed
      </div>

      <div role="main">
         All content inside the main node is processed
```

(continues on next page)

[0] For Sphinx projects, the content of the main node is provided by an intermediate step in the build process, but the HTML components from the node are preserved.

(continued from previous page)

```
        </div>

        <footer>
            This content isn't processed
        </footer>
    </body>
</html>
```

12.1.2 Irrelevant content

If you have content inside the main node that isn't relevant to the page (like navigation items, menus, or search box), make sure to use the correct role or tag for it.

Roles to be ignored:

- `navigation`
- `search`

Tags to be ignored:

- `nav`

Example:

```
<div role="main">
    ...
    <nav role="navigation">
        ...
    </nav>
    ...
</div>
```

12.1.3 Sections

Sections are `h` tags, and sections of the same level should be neighbors. Additionally, sections should have an unique `id` attribute per page (this is used to link to the section). All content below the section, till the new section will be indexed as part of the section. Example:

```
<div role="main">
    <h1 id="section-title">
        Section title
    </h1>
    <p>
        Content to be indexed
    </p>
    <ul>
        <li>This is also part of the section and will be indexed as well</li>
    </ul>

    <h2 id="2">
        This is the start of a new section
    </h2>
```

(continues on next page)

(continued from previous page)

```
   <p>
      ...
   </p>

   ...

   <h1 id="neigbor-section">
      This section is neighbor of "section-title"
   </h1>
   <p>
      ...
   </p>
</div>
```

Sections can be inside till two nested tags (and have nested sections), and its immediate parent can contain the `id` attribute. Note that the section content still needs to be below the `h` tag. Example:

```
<div role="main">
   <div class="section">
      <h1 id="section-title">
         Section title
      </h1>
      <p>
         Content to be indexed
      </p>
      <ul>
         <li>This is also part of the section</li>
      </ul>

      <div class="section">
         <div id="nested-section">
            <h2>
               This is the start of a sub-section
            </h2>
            <p>
               With the h tag within two levels
            </p>
         </div>
      </div>
   </div>
</div>
```

Note: The title of the first section will be the title of the page, falling back to the `title` tag.

12.1.4 Other special nodes

- **Anchors**: If the title of your section contains an anchor, wrap it in a `headerlink` class, so it won't be indexed as part of the title.

```
<h2>
   Section title
   <a class="headerlink" title="Permalink to this headline">¶</a>
</h2>
```

- **Code blocks**: If a code block contains line numbers, wrap them in a `linenos` or `lineno` class, so they won't be indexed as part of the code.

```
<table class="highlighttable">
   <tr>
      <td class="linenos">
         <div class="linenodiv">
            <pre>1 2 3</pre>
         </div>
      </td>

      <td class="code">
         <div class="highlight">
            <pre>First line
Second line
Third line</pre>
         </div>
      </td>
   </tr>
</table>
```

12.2 Overriding the default search

Static sites usually have their own static index, and search results are retrieved via JavaScript. In order for Read the Docs to override the default search as expected, themes from the supported generators must follow these conventions.

Note: Read the Docs will fallback to the original search in case of an error or no results.

12.2.1 Sphinx

Sphinx's basic theme provides the static/searchtools.js file, which initializes search with the `Search.init()` method. Read the Docs overrides the `Search.query` method and makes use of `Search.output.append` to add the results. A simplified example looks like this:

```
var original_search = Search.query;

function search_override(query) {
   var results = fetch_resuls(query);
   if (results) {
```

(continues on next page)

(continued from previous page)

```
      for (var i = 0; i < results.length; i += 1) {
         var result = process_result(results[i]);
         Search.output.append(result);
      }
   } else {
      original_search(query);
   }
}

Search.query = search_override;

$(document).ready(function() {
   Search.init();
});
```

Highlights from results will be in a `span` tag with the `highlighted` class (`This is a <span class="highlighted">result</span>`). If your theme works with the search from the basic theme, it will work with Read the Docs' SSS.

12.2.2 MkDocs

Search on MkDocs is provided by the search plugin, which is included (and activated) by default in MkDocs. The js part of this plugin is included in the templates/search/main.js file, which subscribes to the `keyup` event of the `#mkdocs-search-query` element to call the `doSearch` function (available on MkDocs >= 1.x) on every key press.

Read the Docs overrides the `initSearch` and `doSearch` functions to subscribe to the `keyup` event of the `#mkdocs-search-query` element, and puts the results into the `#mkdocs-search-results` element. A simplified example looks like this:

```
var original_search = doSearch;

function search_override() {
   var query = document.getElementById('mkdocs-search-query').value;
   var search_results = document.getElementById('mkdocs-search-results');

   var results = fetch_resuls(query);
   if (results) {
      empty_results(search_results)
      for (var i = 0; i < results.length; i += 1) {
         var result = process_result(results[i]);
         append_result(result, search_results);
      }
   } else {
      original_search();
   }
}

var init_override = function () {
   var search_input = document.getElementById('mkdocs-search-query');
   search_input.addEventListener('keyup', doSearch);
};
```

(continues on next page)

(continued from previous page)

```
window.doSearch = search_override;
window.initSearch = init_override;

initSearch();
```

Highlights from results will be in a `mark` tag (`This is a <mark>result</mark>`). If your theme works with the search plugin of MkDocs, and defines the `#mkdocs-search-query` and `#mkdocs-search-results` elements, it will work with Read the Docs' SSS.

Note: Since the `templates/search/main.js` file is included after our custom search, it will subscribe to the `keyup` event too, triggering both functions when a key is pressed (but ours should have more precedence). This can be fixed by not including the `search` plugin (you won't be able to fallback to the original search), or by creating a custom plugin to include our search at the end (this should be done by Read the Docs).

12.3 Supporting more themes and static site generators

Currently, Read the Docs supports building documentation from Sphinx and MkDocs. All themes that follow these conventions should work as expected. If you think other generators or other conventions should be supported, or content that should be ignored or have an especial treatment, or if you found an error with our indexing, let us know in our issue tracker.

CHAPTER

THIRTEEN

INTERESTING SETTINGS

13.1 DOCKER_LIMITS

Default: `{'memory': '1g', 'time': 600}`

A dictionary of limits to virtual machines. These limits include:

time An integer representing the total allowed time limit (in seconds) of build processes. This time limit affects the parent process to the virtual machine and will force a virtual machine to die if a build is still running after the allotted time expires.

memory The maximum memory allocated to the virtual machine. If this limit is hit, build processes will be automatically killed. Examples: '200m' for 200MB of total memory, or '2g' for 2GB of total memory.

13.2 SLUMBER_USERNAME

Default: `test`

The username to use when connecting to the Read the Docs API. Used for hitting the API while building the docs.

13.3 SLUMBER_PASSWORD

Default: `test`

The password to use when connecting to the Read the Docs API. Used for hitting the API while building the docs.

13.4 USE_SUBDOMAIN

Default: `False`

Whether to use subdomains in URLs on the site, or the Django-served content. When used in production, this should be `True`, as Nginx will serve this content. During development and other possible deployments, this might be `False`.

13.5 PRODUCTION_DOMAIN

Default: `localhost:8000`

This is the domain that gets linked to throughout the site when used in production. It depends on `USE_SUBDOMAIN`, otherwise it isn't used.

13.6 RTD_INTERSPHINX_URL

Default: `https://readthedocs.org`

This is the domain that is used to fetch the intersphinx inventory file. If not set explicitly this is the `PRODUCTION_DOMAIN`.

13.7 DEFAULT_PRIVACY_LEVEL

Default: `public`

What privacy projects default to having. Generally set to `public`. Also acts as a proxy setting for blocking certain historically insecure options, like serving generated artifacts directly from the media server.

13.8 INDEX_ONLY_LATEST

Default: `None`

In search, only index the `latest` version of a Project.

13.9 PUBLIC_DOMAIN

Default: `None`

A special domain for serving public documentation. If set, public docs will be linked here instead of the `PRODUCTION_DOMAIN`.

13.10 PUBLIC_DOMAIN_USES_HTTPS

Default: `False`

If `True` and `PUBLIC_DOMAIN` is set, that domain will default to serving public documentation over HTTPS. By default, documentation is served over HTTP.

13.11 ALLOW_ADMIN

Default: `True`

Whether to include `django.contrib.admin` in the URL's.

13.12 RTD_BUILD_MEDIA_STORAGE

Default: `readthedocs.builds.storage.BuildMediaFileSystemStorage`

Use this storage class to upload build artifacts to cloud storage (S3, Azure storage). This should be a dotted path to the relevant class (eg. `'path.to.MyBuildMediaStorage'`). Your class should mixin `readthedocs.builds.storage.BuildMediaStorageMixin`.

13.13 ELASTICSEARCH_DSL

Default:

```
{
   'default': {
      'hosts': '127.0.0.1:9200'
   },
}
```

Settings for elasticsearch connection. This settings then pass to elasticsearch-dsl-py.connections.configure

13.14 ES_INDEXES

Default:

```
{
     'project': {
         'name': 'project_index',
         'settings': {'number_of_shards': 5,
                      'number_of_replicas': 0
                      }
     },
     'page': {
         'name': 'page_index',
         'settings': {
             'number_of_shards': 5,
             'number_of_replicas': 0,
         }
     },
 }
```

Define the elasticsearch name and settings of all the index separately. The key is the type of index, like `project` or `page` and the value is another dictionary containing `name` and `settings`. Here the `name` is the index name and the `settings` is used for configuring the particular index.

13.15 ES_TASK_CHUNK_SIZE

Default: `500`

The maximum number of data send to each elasticsearch indexing celery task. This has been used while running `elasticsearch_reindex` management command.

13.16 ES_PAGE_IGNORE_SIGNALS

Default: `False`

This settings is used to determine whether to index each page separately into elasticsearch. If the setting is `True`, each `HTML` page will not be indexed separately but will be indexed by bulk indexing.

13.17 ELASTICSEARCH_DSL_AUTOSYNC

Default: `True`

This setting is used for automatically indexing objects to elasticsearch. `False` by default in development so it is possible to create project and build documentations without having elasticsearch.

CHAPTER

FOURTEEN

TESTING

Before contributing to Read the Docs, make sure your patch passes our test suite and your code style passes our code linting suite.

Read the Docs uses Tox to execute testing and linting procedures. Tox is the only dependency you need to run linting or our test suite, the remainder of our requirements will be installed by Tox into environment specific virtualenv paths. Before testing, make sure you have Tox installed:

```
$ pip install tox
```

To run the full test and lint suite against your changes, simply run Tox. Tox should return without any errors. You can run Tox against all of our environments by running:

```
$ tox
```

By default, tox won't run tests from search, in order to run all test including the search tests, you need to override tox's posargs. If you don't have any additional arguments to pass, you can also set the `TOX_POSARGS` environment variable to an empty string:

```
$ TOX_POSARGS='' tox
```

Note: If you need to override tox's posargs, but you still don't want to run the search tests, you need to include `-m 'not search'` to your command:

```
$ tox -- -m 'not search' -x
```

To target a specific environment:

```
$ tox -e py38
```

The `tox` configuration has the following environments configured. You can target a single environment to limit the test suite:

py38 Run our test suite using Python 3.8

lint Run code linting using Prospector. This currently runs pylint, pyflakes, pep8 and other linting tools.

docs Test documentation compilation with Sphinx.

14.1 Pytest marks

The Read the Docs code base is deployed as three instances:

- Main: where you can see the dashboard.
- Build: where the builds happen.
- Serve/proxito: It is in charge of serving the documentation pages.

Each instance has its own settings. To make sure we test each part as close as possible to its real settings, we use pytest marks. This allow us to run each set of tests with different settings files, or skip some (like search tests):

```
DJANGO_SETTINGS_MODULE=custom.settings.file pytest -m mark
DJANGO_SETTINGS_MODULE=another.settings.file pytest -m "not mark"
```

Current marks are:

- search (tests that require Elastic Search)
- proxito (tests from the serve/proxito instance)

Tests without mark are from the main instance.

14.2 Continuous Integration

The RTD test suite is exercised by Circle CI on every push to our repo at GitHub. You can check out the current build status: https://app.circleci.com/pipelines/github/readthedocs/readthedocs.org

CHAPTER FIFTEEN

ARCHITECTURE

Read the Docs is architected to be highly available. A lot of projects host their documentation with us, so we have built the site so that it shouldn't go down. The load balancer is the only real single point of failure currently. This means mainly that if the network to the load balancer goes down, we have issues.

15.1 Diagram

```
                               +-----------+
                         +-----| Load Bal  |------+
                         |     +-----------+      |
                         |                        |
                    +---------+              +---------+
+-------------+     |         |              |         |    +--------------+
|             |-----| Nginx   |              | Nginx   |----|              |
|  File       |     +---------+              +---------+    |  File        |
|  System     |          |                        |         |  System      |
+-------------+     +---------+  +--------+  +---------+    +--------------+
       |  |         |Gunicorn |  |        |  |Gunicorn |        |   |
       |  +---------|(Django) |--|Postgres|--|(Django) |--------+   |
       |            +---------+  +--------+  +---------+            |
       |                 |                        |                 |
       |                 |                        |                 |
       |                 -----------API------------                 |
       |                             |                              |
       |                             |                              |
       |                     +------------------+                   |
       |                     |                  |                   |
       +---------------------|  Build Server    |-------------------+
                             |                  |
                             +------------------+
```

HTTP ROUTING TABLE